Citation for published version:
Oleinik, E 2004, *Development of an OpenMath-based unit converter to demonstrate the benefits of the*. Computer Science Technical Reports, no. CSBU-2004-12, Department of Computer Science, University of Bath.

Publication date:
2004

Link to publication

©The Author May 2004

University of Bath

Alternative formats
If you require this document in an alternative format, please contact:
openaccess@bath.ac.uk

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

Take down policy
If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.

Download date: 05. May. 2023

Department of Computer Science

Technical Report

Undergraduate Dissertation: Development of an OpenMath-based unit converter to demonstrate the benefits of the newly proposed extension of OpenMath

Eugene Oleinik

Technical Report 2004-12 **May 2004**

ISSN 1740-9497

Copyright ©May 2004 by the authors.

Contact Address:
Department of Computer Science
University of Bath
Bath, BA2 7AY
United Kingdom
URL: http://www.cs.bath.ac.uk

ISSN 1740-9497

Development of an OpenMath-based unit converter to demonstrate the benefits of the newly proposed extension of OpenMath

Eugene Oleinik
BSc in Mathematics & Computing

13th May 2004

Development of an OpenMath-based unit converter to demonstrate the benefits of the newly proposed extension of OpenMath

submitted by Eugene Oleinik

Copyright

Attention is drawn to the fact that copyright of this thesis rests with its author. The Intellectual Property Rights of the products produced as part of the project belong to the University of Bath
(see `http://www.bath.ac.uk/ordinances/#intelprop`).

This copy of the thesis has been supplied on condition that anyone who consults it is understood to recognise that its copyright rests with its author and that no quotation from the thesis and no information derived from it may be published without the prior written consent of the author.

Declaration

This dissertation is submitted to the University of Bath in accordance with the requirements of the degree of Bachelor of Science in the Department of Computer Science. No portion of the work in this dissertation has been submitted in support of an application for any other degree or qualification of this or any other university or institution of learning. Except where specifically acknowledged, it is the work of the author.

Signed: .....

This thesis may be made available for consultation within the University Library and may be photocopied or lent to other libraries for the purposes of consultation.

Signed: .....

Abstract

OpenMath (`http://www.openmath.org`) is an emerging standard for representing mathematical objects with their semantics, allowing them to be exchanged between computer programs, stored in databases, or published on the worldwide web. Content Dictionaries (CDs) are the most important and interesting concept of OpenMath because they define the meaning of objects being transmitted. A CD is a collection of related symbols and their definitions, encoded in XML format. Some of the CDs specify various physical units, such as feet, metres etc., but OpenMath lacks any logical and agreed mechanism for attaching units to quantities using these CDs thus these definitions can not be used for various applications that require working with units. James H. Davenport *in his article "Units and Dimensions in OpenMath" proposes several solutions to this problem which would mean extending definitions of unit CDs. While changes proposed in this article have not yet been accepted by the OpenMath community, we could speed up this process if the usefulness of such extensions could be demonstrated on a real-life application. This project proposes to develop an OpenMath-based unit converter that will take OpenMath descriptions of physical quantities (e.g. "3 feet") and a target system (e.g. "metric") and do the appropriate conversion, using the Content Dictionaries as the source of information. This would mean that someone could write a new CD, e.g. for U.S. units, and the converter would automatically work on these as well. While implementing such a unit converter we could also ensure that all proposed extensions make sense and the extended CDs that have already been created are valid and do not contain errors.*

Acknowledgments

I would like to take this opportunity to express my gratitude to my dissertation advisor Professor James Davenport for formulating this interesting and challenging project as well as for numerous fruitful discussions. I would like to thank my family for encouragement and support during my studies at the University of Bath. I would like to express my warm appreciation and special thanks to Ms. Jacki Hargreaves, my supervisor and colleague at the University of Bath Web team for all her help and understanding that allowed me to successfully combine my undergraduate studies and part-time work.

Contents

Chapter 1

Introduction

OpenMath is an emerging standard for representing mathematical objects with their semantics, allowing them to be exchanged between computer programs, stored in databases, or published on the worldwide web [15]. While the original designers were mainly developers of computer algebra systems, it is now attracting interest from other areas of scientific computation and from many publishers of electronic documents with a significant mathematical content.

Content Dictionaries (or CDs for short) are the most important and the most interesting concept of OpenMath because they define the meaning of the objects being transmitted. A CD is a collection of related symbols and their definitions, encoded in an XML format. OpenMath has a series of Content Dictionaries (CDs) some of which contain descriptions of various physical units, such as feet, metres, etc. It is fair to say that OpenMath lacks any logical and agreed mechanism for attaching units to quantities using these CDs. James Davenport in his article *"Units and Dimensions in OpenMath"* proposes several solutions to this problem (some of which have been adopted from the closely related MathML language [17]). The preferred solution is to use *times* (the symbol representing an n-ary multiplication function) from existing content dictionary *arith1* to link quantities and units in such a way to have the format extension compatible. It would be highly desirable to make proposed system of unit manipulation and processing using OpenMath CDs more effective and useful than they are at the moment.

The major goal of this project is to design an interface and implement a functionality within OpenMath concept to make unit conversions from any given unit system to any other unit system based on existing CDs that contain these units. It is clear that some conversions are straightforward (e.g. convert metres into centimetres) while others are more complicated. The major challenge is to use the recursive structure of newly proposed format of CDs that allows the units to be defined in terms of other units which are in turn defined via some other units. Another challenge is to perform reverse conversions, i.e if the *base* unit system is International System of Units (SI) reverse conversions to imperial

system (e.g. from metres into miles) become more complicated.

One of the major requirements to the mechanism of unit conversion is to make sure that the required conversion is legal, that is one type of unit can not be accidentally converted into another type of unit (e.g. to convert pascals into metres). The system must also be able to convert between the *composed* units (composed means that prefixes like *milli* can be attached to units). In addition, some measures do not have their equivalents across all unit systems so this must also be taken into account and dealt with appropriately.

There is also a need to rationalize the system of unit conversion to ensure the consistency of definitions and formats of unit CDs. The challenging question arisen at the stage of formulation of the project is the following: is it necessary to interpret all the OpenMath Symbols (OMSs) or maybe, or is it better to define internally those OMSs which are specified in *relation1* and *arith1* CDs in order to reduce the amount of processing needed for such a converter?

After careful analysis of the project goals, the following functionalities of OpenMath unit converter has been formulated for implementation:

1. To get a list of available unit systems from the configured location;
2. To download CDs and all other necessary files needed for conversion;
3. To work with any number of unit systems and easily add new unit systems;
4. To interpret CDs (parse the format and understand the data);
5. To display units available for conversion;
6. To get user input and process it;
7. To produce needed conversions;
8. To have a user-friendly interface and be intuitive to use;
9. To be easy to install and configure;
10. To be robust and have a good error checking system.

Choice of programming language for implementing OpenMath unit converter depends on the ability to parse XML-like data (using additional modules if needed) and to produce adequate user-friendly interface. The simplicity of working with XML data is one of the requirements for the successful design of unit converter.

Taking into account all these requirements we implemented the OpenMath unit converter based on content dictionaries using Perl programming language. As a result we were managed to demonstrate successfully that the proposed extension to OpenMath that aims at attaching units to quantities does work extremely well and will greatly benefit the community by creating novel and useful system tools within OpenMath framework.

The outline of this thesis is the following. In chapter 2 we present the background information including the concepts of OpenMath, Extensible Markup Language (XML), content dictionaries (CDs), proposed extensions of CDs including unit processing, existing unit converters outside OpenMath. Chapter 3 is the central part of this thesis which the project implementation including system requirements and specifications, and resources used. Chapter 5 describes the unit converter testing. Chapter 6 presents conclusions including future work and improvements. Appendix A contains development notes and appendix B is the user manual of the developed unit converter.

Chapter 2

Background information

This chapter includes traditional literature survey as well as the information that is published on WWW. The OpenMath concepts are nicely described at the project web site `http://www.openmath.org`. My particular focus was to understand and become fluent with the novel concepts of OpenMath, XML and other related subjects. They include:

- General concepts of OpenMath,
- XML & Document Object Model (DOM) and their relationship with OpenMath,
- OpenMath Architecture and Objects,
- Content Dictionaries (CDs),
- Signature Dictionaries based on Simple Type Systems (STSs),
- Proposed extension of CD format.

Before implementing the project I decided to make a comparative review of already existing unit converters. They are also reviewed in this chapter.

2.1 OpenMath

OpenMath is a standard for representing mathematical data in as unambiguous way as possible. It can be used to exchange mathematical objects between software packages or via email, or as a persistent data format in a database. It is tightly focused on representing semantic information and is not intended to be used directly for presentation, although tools exist to facilitate this.

The original motivation for OpenMath came from the Computer Algebra community. Computer Algebra packages were getting bigger and more unwieldy. Therefore, it seemed reasonable to adopt a generic "plug and play" architecture to allow specialized programs to be used from general purpose environments. There were plenty of mechanisms for connecting software components together, but the common format for representing the underlying data objects was absent. It quickly became clear that any standard has to be vendor-neutral and that objects encoded in OpenMath should not be too verbose. This has led to the design implemented within on-going OpenMath efforts.

Since 1997 OpenMath development has been partially funded by the European Union under a multimedia European Strategic Programme for Research in Information Technologies (ESPRIT) project. In 1998, the W3C[1] produced it's first recommendation for the Extensible Markup Language (XML), intended to be a universal format for representing structured information on the world-wide web. It was swiftly followed by the first MathML recommendation which is an XML application oriented mainly towards presentation (i.e. rendering) of mathematical expressions [5]. The formal definition of OpenMath is contained within The OpenMath Standard and its accompanying documents (see `http://www.openmath.org/cocoon/openmath/standard/index.html`).

Both MathML and OpenMath are XML applications, these two systems can be thought of as analogous to each other, but there is a constant feeling of tension between OpenMath and MathML communities. Work is being done to make both standards compatible/compliant to each other but it is very difficult, many arguments and questions arise during discussions mainly because the philosophy of those two standards is so different. MathML was primarily aimed at solving a task of rendering without taking into account the actual meaning of the data being represented. On the other hand, OpenMath concentrates on giving every object a sensible meaning and compensates this by complicating the rendering side. The major effort of new European project, Thematic Network, is to align OpenMath and MathML and to produce a Reduce-based OpenMath/MathML translator.

2.2 XML and its relationship with OpenMath

XML (eXtensible Markup Language) is a W3C-endorsed standard for text document markup [18]. It defines a generic syntax used to mark up data with simple, human-readable tags. Data are included in XML as strings of text and are surrounded by text markup which describes the data. XML's basic unit of data and markup is called an *element*. XML is a metamarkup language, i.e. it does not have a fixed set of tags and elements that will work for everybody in all areas of interest for all time. Instead, XML allows developers and writers to define the sets of elements they need. Those sets are often named as XML *applications* (i.e applications of XML to a specific area of interest).

[1] World Wide Web Consortium, an organisation responsible for developing and maintaining many data format standards, e.g. XML, HTML, CSS.

Although XML is quite flexible in the elements it allows to be defined, it is quite strict in many other respects. W3C XML specifications provide a grammar (syntax) for XML documents that says where tags may be placed, what they must look like, which element names are legal, how attributes are attached to elements, and so forth. This "strictness" of grammar allows creation of a standard XML parser that will be able to process any XML document no matter what its *application* is.[21]

The markup in an XML document describes the structure of the document. It lets you see which elements are associated with which other elements. In a well-designed XML document, the markup also describes its semantics. e.g. the markup can indicate whether an element is a date or a person or a bar code. It is also important to note that XML has a hierarchical structure since it's roots date back to the time when hierarchical databases were very popular. Importantly, in a well-designed XML applications the markup allows a flexible display of the document without strict specification of the format of the output [21].

The markup permitted in a particular XML *application* can be documented in a *schema*. Validity of any XML *application* document can be verified by checking whether it matches the *application* schema. The most popular *schema* at the moment is Document Type Definition (DTD). DTDs are used for HTML, XHTML, MathML, OpenMath and other systems[21].

The XML parser that was mentioned above is responsible for structuring document's individual elements, attributes and other pieces. The parser passes the contents of the XML document bit by bit. Individual XML applications normally dictate more precise rules about exactly which elements and attributes are allowed and where. Some of those rules are specified using *schema* as described earlier, e.g. using DTD.

Apart from thinking of XML as a plain-text set of strings, there are several Application Programming Interfaces (APIs) created to access and manipulate XML data. Two most popular APIs are DOM and SAX.

DOM stands for *D*ocument *O*bject *M*odel (derived by W3C) which describes the method of processing an XML document as a tree of nodes, every single data item being thought of as a node in the document tree. DOM maps an XML document into an internal tree structure, then allow an application to navigate that tree. DOM navigation methods are used to move around the tree and allows for easy tracking of interrelationships between the elements [16].

SAX stands for *S*imple *A*PI for *X*ML (SAX) and it is event-based API. In contrast to tree-based API such as DOM, SAX reports parsing events (such as the start and end of elements) directly to the application through callbacks, and does not usually build an internal tree. The application implements handlers to deal with the different events, much like handling events in a graphical user interface, i.e SAX triggers an event and the parser calls the corresponding handler function in the code to handle the event [11].

2.3 OpenMath Architecture and Objects

The OpenMath representation of a mathematical structure is referred to as *OpenMath object*. This is an abstract structure which is represented concretely via an *OpenMath encoding*. These encoded objects are what an OpenMath application would read and write. In practice the OpenMath objects themselves almost never exist, except on paper. The advantage of this is that OpenMath is not tied to any underlying mechanism in contrast to previous practice when functional, SGML and binary encodings were used. The current favorite mode of OpenMath is to use XML to describe OpenMath objects (even though the XML representation is an encoding itself) [5].

OpenMath application object is viewed as a "tree" by software applications that do not understand Content Dictionaries. The conversion of an OpenMath object to/from the internal representation in a software application is performed by an interface program called Phrasebook. The translation is governed by the Content Dictionaries and the specifics of the application. It is envisioned that a software application dealing with a specific area of mathematics declares which Content Dictionaries it understands. As a consequence, it is expected that the Phrasebook of the application is able to translate OpenMath objects built using symbols from these Content Dictionaries to/from the internal mathematical objects of the application [22].

OpenMath objects are the elements of a labeled tree whose leaves are the basic objects including integers, floating-point numbers, character strings, byte arrays, variables and symbols. Symbols are the most interesting OpenMath objects since they consist of a name and a reference to a definition in an external document called Content Dictionary. OMS stands for OpenMath Symbol and using XML notation the following line <OMS name="sin" cd="transc1" /> represents the usual sin function as defined in the CD "transc1". A basic OpenMath object has the following representation:

```
<OMOBJ>
   <OMS name=''sin'' cd=''transc1'' />
</OMOBJ>
```

OpenMath objects can be built up recursively in a number of ways. The simplest is the function application. For example, the expression sin(x) can be represented by the XML as

```
<OMOBJ>
<OMA>
   <OMS name=''sin'' cd=''transc1'' />
   <OMV name=''x'' />
</OMA>
</OMOBJ>
```

where OMV introduces a variable and OMA is the application element. Another straightforward method is attribution which as the name suggests can be used to add additional information to an object without altering its fundamental meaning [5].

2.4 Content Dictionaries

Content Dictionaries (CDs) are the most important and interesting concept of OpenMath because they define the meaning of objects being transmitted. A CD is a collection of related symbols and their definitions, encoded in XML format. Defining the meaning of a symbol is not a trivial task and even referring to well-known references can be fraught with pitfalls. Formal definitions and properties can be very useful but time-consuming to produce and verbose, not to mention difficult to get right. A symbol definition in OpenMath CD consists of the following pieces of information:

- the symbol name
- a description in plain text
- set of symbol's properties in plain text - Commented Mathematical Properties (CMPs), optional
- set of symbol's properties encoded in OpenMath - Formal Mathematical Properties (FMPs), optional
- one or more examples of its use - encoded in OpenMath, optional

In practice, the CMPs and FMPs can come as pairs, and often serve in the place of examples [5]. Here is a simple instance of a CD definition:

```
<CDDefinition>
<Name> mile </Name>
<Description>This symbol represents the measure of one mile.
This is the standard imperial measure for distance.</Description>
</CDDefinition>
```

CDs usually consist of related symbols, and for the purpose of convenience the collections of related CDs can be grouped together. For example, *CD Groups* (*units_ imperial1.ocd*) defines symbols to represent standard measures.

Given the evolutionary nature of mathematics, it is clear that the set of CDs will be constantly growing and will never be complete. Therefore the CD format is constantly evolving to include the most useful information about symbols [5]. We will discuss this process in the section 2.7 on page 10.

2.5 Signature Dictionaries

It is possible to associate an extra type information with CD. There are many type systems available, but simple signatures are preferred to be encoded using the Small Type System as described in[3]. Types are associated with OpenMath objects using attribution. This is done by (i) creating a CD that specifies constructors of the Type System, (ii) building OpenMath objects representing these types. Here is an example of the signature for the metre, the standard SI unit of length:

```
<Signature name="metre" >
<OMOBJ>
    <OMS cd="dimensions1" name="length"/>
</OMOBJ>
</Signature>
```

The most interesting areas of signature usage related to this project are the *units* for defining type of each unit (e.g. length) and prefixes for defining type of each prefix.

2.6 Location of Content Dictionaries

All Content and Signature Dictionaries are stored at the OpenMath official website (`http://www.openmath.org`) in section "Content Dictionaries". CDs are sorted into subsections according to their importance and readiness for the OpenMath world. There are four subsections in the Content Dictionaries section:

1. *Core* - this part contains a core set of CD Groups that is required to understand OpenMath syntax and a MathML CD group which provides compatibility with the Content markup of MathML 2.
2. *Public* - this part contains CDs that have been reviewed and endorsed by OpenMath community (these CDs have stable "official" status). It is useful to note that at the present moment this subsection also has only key CDs: they define symbols used in OpenMath constructs (the meta CD Group), in the signature files (for the simple type system for OpenMath), and for OpenMath error handling.
3. *Extra* subsection has more interesting content. It has a full set of OpenMath Content Dictionaries (CD) and Signature Files that are available from `http://www.openmath.org`. This section contains not only key CDs from *Public* and *Core* subsections but also other dictionaries which have not yet been assigned status "public" (they include the old versions of CDs

for defining units). The dictionaries from *Extra* subsection are constantly evolving: some of the CDs have been updated and newer versions (stored in the *Contributed* subsection) have been proposed.

4. *Contributed* subsection contains all the newly proposed CDs that have been added using the online submission form. CDs in this section have not yet been reviewed. The good practice is always to check that there are no errors in these CDs. For example, it was discovered in the course of this project that there is no link between metric and imperial pressures. It is always recommended to create local copies of the CDs downloaded from this section for subsequent revision and improvements.

The final version of the unit converter implemented in the course of this work will reference a central official location of CDs but special checks must be performed to make sure that all the necessary CDs migrated from *Contributed* to *Extra* subsection.

For the purpose of the current project the following CD files have been used:

- Arithmetic and operation CDs from Extra section:
  `arith1.ocd & relation1.ocd`
- All "units_*" files found in *Contributed* subsection (including STS files). Currently, they include:
  `units_imperial1.ocd & units_imperial1.sts`
  `units_metric1.ocd & units_metric1.sts`
  `units_time1.ocd & units_time1.sts`
  `units_siprefix.ocd & units_siprefix.sts`

2.7 Proposed extension of CD format

One might ask a question: "We have these wonderful definitions of symbols using CDs but can we do anything useful with them?". For example, take unit CDs (CD defining different unit measures), these describe various measures like feet, metres, litres, pints, etc. How can we do anything useful with them using CDs? How can conversion between different units be actually performed? It is fair to say that OpenMath lacks any logical and agreed mechanism for attaching units to quantities using these CDs. James H. Davenport in his article *"Units and Dimensions in OpenMath"* proposes several solutions to this problem (some of which have been adopted from the closely related MathML language [17]). The preferred solution is discussed in detail below.

2.7.1 Representation of units

It is necessary to express all unit systems in terms of a metric system. Using *times* from relation1 CD express the basic measure in terms of a metric system. For example foot: express foot (standard of length in imperial) in terms of metres. All other measures in the particular system are represented in terms of the standard measure. For example, mile would be represented in terms of feet as:

```
<CDDefinition>
<Name> mile </Name>
<Description>This symbol represents the measure of one (land,
or statute) mile.  This is a standard imperial measure for
distance, defined in terms of the foot.</Description>

<CMP> 1 mile = 5280 feet </CMP>

<FMP><OMOBJ>
    <OMA>
        <OMS name="eq" cd="relation1"/>
        <OMA>
            <OMS name="times" cd="arith1"/>
            <OMI> 1 </OMI>
            <OMS name="mile" cd="units_imperial1"/>
        </OMA>
        <OMA>
            <OMS name="times" cd="arith1"/>
            <OMI> 5280 </OMI>
            <OMS name="foot" cd="units_imperial1"/>
        </OMA>
    </OMA>
</OMOBJ></FMP>

</CDDefinition>
```

Updated CDs that comply with this proposal have been deposited at Contributed section of Content Dictionaries at `http://www.openmath.org`. They are:

```
units_metric1.ocd
units_imperial1.ocd
units_time1.ocd
```

We recap again that every unit, unless it's a *base SI* unit, must be defined in terms of some other unit. Because SI system is agreed to be metric, "units_metric1" CD is the minimum CD necessary to define any units. All other systems may

define their units using "units_metric1" CD. Following this idea it was suggested in [4] to rename "units_metric1" into "units_si1".

2.7.2 Attribution

The next step is to add quantities to units. For example, we would like to represent 2 grammes in OpenMath as follows

Number 2 in OpenMath is:

```
<OMI> 2 </OMI>
```

Gramme in OpenMath is:

```
<OMS name=''gramme'' cd=''units_metric1'' />
```

We can use *times* from *arith1* to bound them together like so:

```
<OMA>
   <OMS name=''times'' cd=''arith1'' />
   <OMI> 2 </OMI>
   <OMS name=''gramme'' cd=''units_metric1'' />
</OMA>
```

2.7.3 Unit prefixing

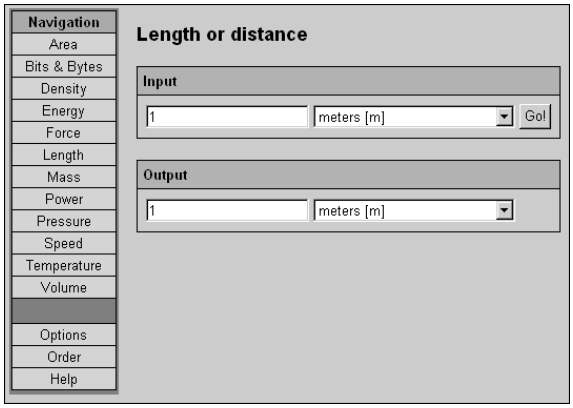
The next issue to address is to deal with units that are composed of prefixes. Creating separate CD entries for them would cause many inconsistencies and be very verbose: each unit would acquire 20 variants. Instead, it was proposed to create a special operator such as <OMS name="prefix" cd="units_ops1"> whose signature is given by:

```
<Signature name=''prefix''>
<OMOBJ>
<OMA>
   <OMS name=''mapsto'' cd=''sts''>
   <OMS name=''unit_prefix'' cd=''units_sts''>
   <OMV name=''dimension''>
   <OMV name=''dimension''>
</OMA>
</OMOBJ>
</Signature>
```

Following the rules of the Small Type System[3], the first argument must have type unit_prefix (i.e. must be a prefix), the second argument can be anything ("something of an unknown dimension") but must return something of the same kind ("the same dimension"). CD ("units_siprefix.ocd") and STS ("units_siprefix.sts") files for unit prefixing have been created according to this proposal and can be found in the Contributed section of Content Dictionaries at `http://www.openmath.org`.

2.8 Existing unit converters

Prior to implementing this project, I decided to review existing unit converters in order to understand state of the art methodologies and technical approaches. There are a number of Unit converters available on the Internet, many of them execute conversions online using web interface. One such converter hosted by Digital Dutch can be found at: `http://www.digitaldutch.com/unitconverter/`. The screenshot of this converter is shown below:

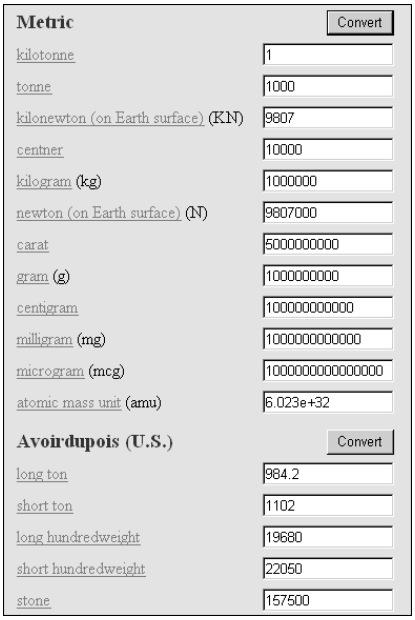


The interface of digital dutch converter is relatively simple: there is a menu to choose the type of measure for conversion (e.g. lengths, pressures). Once the type of measure has been chosen, new window appears with two list boxes: input and output unit and a quantity of input unit. I immediately spotted a problem seems to be a common drawback of all unit converters that I inspected: they do not use official definitions of unit values in relation to each other. Therefore, these systems are not easy to extend when a new unit type or unit system is added.

I noticed that the interface would be much more practical if all types of units were converted from one page rather than navigating through different pages using a menu on the left. This particular layout of input boxes does not allow an easy handling of this situation. Another shortcoming of this system is that it presents no options to choose the prefix for the units. While in most cases it is convenient to choose kilometres from a single list box, it is not always a good

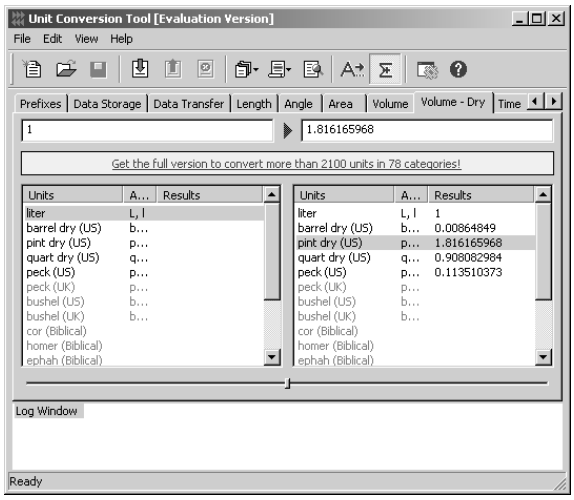
idea. This is because every length unit has all prefix variants in a list box which produces too many entries to be selected from in the list-box. This means that it is very difficult to find quickly the necessary unit. As I mentioned above, I found several similar systems both web-based and standalone running on MS Windows platforms.

Another approach for constructing unit-converter was found at [14]. The screenshot is shown below:



This web-based system converts different weight units and is implemented as a Java script. All the units are listed on a single page separately. Once a number has been entered in one of the units and *Convert* button has been pressed, the rest of the unit boxes show the converted values. This interface is good when comparing different unit values, but it is impractical because you can only convert one type of units from one page. In addition, the units are hard to find and the prefixes are not split up.

An interesting converter has been found on the same page as a link to a downloadable Windows executable (shareware):



I immediately spotted interesting aspects of this converter that I would like to implement in my own project. Although it is not based on OpenMath CDs but has some sort of semantic system that defines some units in terms of other units. Moreover, it has some sort of prefix definition. This product has some other unique features like the ability to accept complex expressions like formulae in the input boxes and ability to add/modify units. However, the interface of the converter is not user-friendly: some bookmarks for selecting measure types do not fit into the window, and the user must use arrows for scroll up and down. This is an important drawback since the user is not able to see all available measures at once and has to click many graphical objects to achieve what he/she wants.

OpenMath-based unit converter - project plan				
Key	Task Name	Work (hours)	Duration (days)	% Complete
1	OpenMath-based unit converter - project plan	357,8	1,4	100
32	Project write-up	227,7	1,4	100
82	Initial Project Plan	1,1	0,0	100
35	Project proposal	12,5	1,0	100
36	Initial version	8,0	1,0	100
37	Check that supervisor is happy with proposal	0,5	0,0	100
38	Final version	4,0	0,2	100
8	Literature Review (theory)	62,0	1,4	100
9	Compile a list of relevant literature including XML-related literature	1,0	0,0	100
51	Study XML format	24,0	1,4	100
52	Study "XML in a nutshell" book	16,0	1,4	100
53	Produce an overview of XML for write-up	8,0	1,0	100
39	Study OpenMath format (parts relaing to CDs & STSs)	20,0	0,4	100
40	View format v1 specs	1,0	0,0	100
41	View format v1.1 specs	1,0	0,0	100
43	View format v2 draft specs	4,0	0,2	100
59	Study further literature on OpenMath	6,0	0,1	100
60	Read "Writing CDs" paper by JHD	2,0	0,1	100
61	Read "A Small OpenMath Type System" paper by JHD	2,0	0,1	100

OpenMath-based unit converter - project plan				
63	Read "OpenMath: An Overview" paper by M. Dewar	2,0	0,1	100
58	Produce an overview of OpenMath for the writeup	8,0	0,4	100
54	Study proposed extension of OpenMath	9,0	0,3	100
44	Read "Units & Dimensions in OpenMath" paper by JHD	4,0	0,2	100
45	Comment on this paper in write-up in details	5,0	0,3	100
46	Study the structure of the OpenMath CD directory and comment for write-up	8,0	1,0	100
64	Existing products	6,5	0,3	100
65	Identify existing products	1,5	0,1	100
66	Produce a review of existing products including pros. vs cons. of particular features	5,0	0,3	100
67	System requirements	10,0	1,0	100
69	Produce a list of requirements	2,0	0,1	100
70	Discuss each requirement in detail for the write-up	8,0	1,0	100
7	Resources	11,1	0,3	100
48	Compile a list of available XML parsers and classify them	3,0	0,1	100
49	Comment on found parsers in write-up	5,0	0,3	100
71	Make a decision on Programming language and discuss it in the write-up	0,3	0,0	100
47	Make a decision on what parser to use and reason it in the write-up	2,0	0,1	100
11	Make a decision on extra resources	0,8	0,0	100
12	Discuss operating system and web server decisions	0,5	0,0	100
13	Discuss project control tools	0,3	0,0	100

OpenMath-based unit converter - project plan			
15 Final project plan	3,5	0,1	100
5 Discuss the planning strategy	0,5	0,0	100
72 Produce final version of project plan	3,0	0,1	100
73 System specifications	79,0	1,4	100
74 Write about startup data processing	7,0	0,3	100
75 Document STS file processing	8,5	1,0	100
76 User interface	11,0	0,3	100
77 Discuss possible UI versions and make a final decision	6,0	0,3	100
79 Describe UI generation	5,0	0,3	100
80 Describe input parsing	1,5	0,1	100
81 Conversion system	51,0	1,4	100
83 Describe processing prefixes	8,0	1,0	100
84 Units processing	40,0	1,4	100
108 Describe parsing function	8,0	1,0	100
109 Describe function that finds an object definition	8,0	1,0	100
110 Describe operation emulator function	8,0	1,0	100
111 Describe unit converter	16,0	1,4	100
85 Write about integration with an interface	3,0	0,1	100
86 Testing	28,0	0,4	100
117 Fixing bugs in current CDs	7,0	0,3	100

OpenMath-based unit converter - project plan			
Preparation of new custom CDs	8,0	0,4	100
Requirements testing	7,0	0,3	100
Testing report for write-up	6,0	0,3	100
Conclusion write-up	10,0	0,1	100
Drawbacks of the system	3,5	0,1	100
Improvements that could be done	3,5	0,1	100
Overall conclusions	3,0	0,1	100
Produce user manual	4,0	0,2	100
Development	126,1	1,1	100
Install web server, Perl and XML parser	3,0	0,1	100
Check that installed software is functioning as required	2,0	0,1	100
Produce development notes on preparations for developing	4,0	0,2	100
Produce development notes on getting familiar with Content Dictionaries	2,0	0,1	100
Do some excercises	20,1	1,1	100
Read about how to parse XML data	5,0	0,3	100
Do some exercises - parsing XML data	5,0	0,3	100
Do some exercises - processing OpenMath data	10,1	1,1	100
Implement startup processing	6,0	0,3	100
User Interface	13,5	1,0	100
Do startup processing testing	0,5	0,0	100

OpenMath-based unit converter - project plan				
57	Produce initial version of user interface script	8,0	1,0	100
56	Do interface testing and add anything found to the bug tracker	1,0	0,0	100
94	Correct any bugs found in interface generator	4,0	0,2	100
96	User input parsing	11,5	0,3	100
98	Produce an initial version of input parser	7,0	0,3	100
99	Do testing, any bugs found should be added to bug tracker	0,5	0,0	100
100	Finalise input parser	4,0	0,2	100
97	Conversion system	50,0	1,0	100
101	Produce forward non-base to base primitive converter	7,0	0,3	100
103	Test previous task and debug	1,0	0,0	100
119	Produce forward non-base to base non-primitive converter	3,0	0,1	100
120	Test previous task and debug	1,0	0,0	100
121	Produce backward base to non-base primitive converter	9,0	1,0	100
118	Test previous task and debug	1,0	0,0	100
123	Produce backward base to non-base non-primitive converter	4,0	0,2	100
124	Test previous task and debug	1,0	0,0	100
125	Produce non-base to non-base primitive converter	2,0	0,1	100
126	Test previous task and debug	1,0	0,0	100
122	Produce non-base to non-base non-primitive converter	8,0	1,0	100
127	Test previous task and debug	1,0	0,0	100

OpenMath-based unit converter - project plan			
Produce composite converter	4,0	0,2	100
Test previous task and debug	2,0	0,1	100
Full unit converter system testing and debugging	5,0	0,3	100
Add any deviations from the plan to the development notes section	8,0	1,0	100
Do final testing	2,0	0,1	100
Correct any bugs found after final testing	4,0	0,2	100

Chapter 3

Project implementation

This chapter describes the actual work done on developing OpenMath unit converter. The project implementation plan included

1. formulation of system requirements,
2. researching the best programming resources available for implementing the project and making appropriate decisions,
3. working on detailed system specifications
4. actual project implementation in the form of developing, testing and documenting program modules that comprise the OpenMath unit converter.

It is worth mentioning that this document was created in the course of the project and reflects the actual dynamics of the process. My main strategy was to work in multitasking environment, that is to work simultaneously on several independent aspects of the project that require considerable time investment. The consistency of the goals has been achieved by keeping track of project stages using project planning and bug tracking tools. This approach allowed to work persistently on program coding, benchmarking and error testing, together with constant documenting of the goals achieved and problems discovered.

3.1 System requirements

The major requirements for my OpenMath unit converter were enlisted in introduction. Here we discuss these points in detail.

A. To get a list of available unit systems from the configured location

System must be able to start up by having one piece of data - the location of CDs and their corresponding STSs. The system should then be able to detect which of those CDs represent unit definitions, check that signature files for those CDs are available as well to check that all the other necessary supplementary CDs are present (e.g a CD containing prefix information).

B. To get a list of available unit systems from the configured location

Required CDs will then be loaded every time before conversion is begins. System must be able to work with local addresses only. It was initially thought to work with remote URLs but this would create a less reliable system due to directory listing, connection and URL redirect problems. References to loaded files should then be stored at some predefined storage location in memory.

C. To work with any number of unit systems and easily add new unit systems

The system must be able to work with any number of unit systems, automatically update the new units and include them in the list of available units for conversion. If a new CD with a new unit system has been created, new CD along with the corresponding STS file is placed in a central location where all those unit CDs are stored.

D. To interpret CDs (parse the format and understand the data)

The converter must be able to parse XML data contained in CD and STS files and extract needed information. It must understand the CD structure and be able to process composite statements (definitions) i.e. be recursive.

E. To display units/prefixes available for conversion

After reading appropriate CDs the system must list these units that are available for conversion. STS information should be used to group various units from different systems into types (e.g. length, volume, pressure) and to list available prefixes. The system must be able to deal appropriately with the units that have no equivalent in other systems.

F. To get user input and process it

User should be able to choose the required conversion, enter the values, choose prefixes and submit data for conversion. The input should be processed and checked for validity:

- Are the input data valid? (e.g. are the value inputs "pure" numbers?)
- does the prefix chosen exist?
- do the units exist?
- is the conversion valid? (e.g. are both input and output units of the same type?)
- is it possible to do the conversion?

G. To produce needed conversions

The system must then do the conversion. The conversion has to be split into several tasks:
To split value that user provided for the quantity of input unit, process the prefix information to get the necessary coefficients, use recursive approach to do the *base conversion* (to convert without taking into account prefixes and user-supplied quantities). The result of the base conversion is then multiplied by the prefix values (output prefix actually means divide) and multiplied by the quantity that user supplied.

Before performing actual *base conversion*, the conversion must be classified in order to determine how to convert and how far the recursive level goes. These aspects of classification are considered below when discussing the technical details of project implementation.

H. To have a user-friendly interface and be intuitive to use

The unit converter is not a terribly complicated mathematical calculation but just matter of convenience. Therefore the interface must be as user-friendly as possible, be intuitive and easy to use. The major emphasis must be made on convenient way of displaying possible conversions and choosing what user wants to convert.

I. To be easy to install and configure

The system must be relatively easy to install. Assuming that the operating system has all pre-requisite modules and libraries installed, all what is needed to

install the system is to unpack the converter archive file and set the appropriate permissions on executable files (this will be outlined in section 3.2.4 on page 21).

J. To be robust and have a good error checking system

The system must be stable and not crash irregardless of user errors, errors containing in unit CDs, STS or a simply broken internet connection.

3.2 Resources

XML is all about structuring the data in the way which is required for using tree structures. *XML parser* is the prerequisite for reading XML documents. Parser simply recovers the data by passing parts of the document bit by bit. There are many parsers freely available for most popular programming languages.

As we have already discussed in 2.2, there are three types of XML parsing methods:

- Usual XML parsing (as a text)
- DOM parsing
- SAX parsing

Here we review the most popular products and resources available for each method of parsing.

3.2.1 XML parsing as a text

Expat
Implemented as C++ library, but there are numerous extensions for Perl, php & Python.
Expat is a library written in C++ for parsing XML documents. It is used in XML parser for Mozilla web tools project, Perl's XML::Parser and other open-source XML parsers. It is very fast (although the speed is not an important aspect of the current project) and sets a very high standard for reliability, robustness and correctness. It is easy to use: you simply register a call-back (or handler) functions with the parser and then start feeding the document. As the parser recognizes parts of the document, it will call the appropriate handler for the particular part.[8]

XML::Parser
As mentioned above, this is an extension to Expat C++ library. This comes with a standard Perl installation.[12]

Libxml2
This is a C library developed for the Gnome project (although it can be used outside the Gnome platform). Although the library is written in C, there are several extensions that allow the library to be used in other languages (mainly php and Perl).[19]

3.2.2 DOM parsing

Xerces
There are several Xerces (named after the Xerces Blue butterfly) libraries under this name which are very strongly promoted by Apache project. Fully validating parsers are available for both C++ and Java. Perl wrapper is also available.[20]

- **Xerces C++**
  Xerces C++ is a validating XML parser which makes it easy to give an application the ability to read and write XML data. A shared library is provided for parsing, generating, manipulating and validating XML documents.
- **Xerces2 Java parser**
  Xerces2 is a fully conforming XML Schema processor. It also provides an experimental implementation of DOM Level 3 Core.
- **XML::Xerces**
  XML::Xerces is the Perl API to the Xerces XML parser. It provides a validating XML parser. Classes are provided for parsing, generating, manipulating and validating XML documents. The module is faithful to DOM Level 3.

XML::DOM
This is a Perl module built on top of XML::Parser which parses XML documents using DOM Level 1.

php5 DOM extension
This an extension based on the Libxml2 library that offers an implementation of XML DOM Level 3. This module is included by default in the standard php installation.[13]

PyXML
PyXML package is a collection of libraries to process XML with Python. It contains a validating parser and a fully compliant DOM Level 2 implementation. [10]

Java API for XML Processing (JAXP)
The Java API for XML Processing (JAXP) is for processing XML data using applications written in the Java programming language. JAXP leverages the parser standards SAX (Simple API for XML Parsing) and DOM (Document

Object Model) so that you can choose to parse your data as a stream of events or to build an object representation of it.[9]

RIACA OpenMath Library
The OpenMath Library project provides a possibility to use OpenMath modules written in Java. Includes an OpenMath DOM reader.[6]

3.2.3 SAX parsing

Java API for XML Processing (JAXP)
The Java API for XML Processing (JAXP) is for processing XML data using applications written in the Java programming language. JAXP leverages the parser standards SAX (Simple API for XML Parsing) and DOM (Document Object Model) so that you can choose to parse your data as a stream of events or to build an object representation of it.[9]

XML::Parser
XML::Parser operates as a SAX parser by default. Handler functions are assigned to the various events by passing a hash with the names of these handlers and references to the function calling XML::Parser class *new* method.

3.2.4 Resource decisions

It is clear that for each parsing method there are parsers available for almost every programming languages. Therefore, it is important to make a decision based on effectiveness of the required operations.

With two powerful extensions of "plain-text" XML parsing, it is wise to use these extensions since this would make a project development easier and faster. The main question is which API should be used for my project: SAX or DOM?

Many experts say that SAX is a lot easier to work with. However, there are no mechanisms for navigating through the document data, moving up and down the nodes since there is no node tree in SAX. The only possibility to accomplish this is to wait until the item you want to navigate to is passed to you by the SAX parser. Mainly due to this reason I decided to use DOM parsing in my project. Thinking of CDs as a tree-structured documents is the best way to deal with them.

The only disadvantage that DOM parsing presents is its speed. DOM parser builds full document tree in memory, and this causes an additional overhead of keeping the tree structure in memory. My project is not oriented at providing a server solution that will be used by hundreds of people at one instance, nor does the project that would involve dealing with big XML trees. Therefore this disadvantage is of minor importance.

RIACA OpenMath Library looks like the best option for DOM parsing as it is targeted for OpenMath development. However, after inspection I came to the conclusion that the use of the library does not provide any significant advantages compared to standard XML::DOM or php DOM extensions. Since both libraries are standard extensions of Perl this decision makes the technical part of the project much more easier. Perl is the program language of choice for work with large databases as well as it is exceptionally efficient in interfacing with HTML, XML, and other mark-up languages. (full list of Perl subjects that I mastered is given in the section A.1 on page 56). Perl also has another advantage - it has very powerful text processing capabilities which I mastered not only during the work on this project but also during all my previous work as web developer at the University web team. Therefore I decide to use Perl with XML::DOM module for my project. This module has been around for a while, is very stable and is widely used by programming community. It also extensively documented in many publications. In particular, this module is covered in detail on the book ("Perl Black Book", Steven Holzner, Coriolis, 2001) that I use as Perl desktop reference in my everyday practice.

Because I have chosen to use Perl as my programming language, best user interaction can be achieved by creating a web-based system. This means that apart from Perl interpreter and XML::DOM module, a web server is also required. Unlike XML parsers, the choice of a web server for this project is not an important issue, since nearly any web server can be used. I decided to develop UNIX based system. Therefore, the obvious web server of choice is Apache since it is included in all standard distributions of UNIX including Linux, FreeBSD and Mac OS X. In addition, I also envisioned the necessity to install OpenMath unit converter on MS Windows type PCs. In this case I made a choice for the Small HTTP Server (`http://home.lanck.net/mf/srv/`) because of its small disk, memory and CPU consumption it can be very easily and quickly installed by nonspecialist.

For actual project planning and control I used two resources:

1. jxProject tool (`http://www.jxproject.com`) for project planning;
2. Mantis bug tracking tool (`http://www.mantisbt.org`) for keeping track of all bugs as they appear and for subsequent project testing.

3.3 System specification and implementation

I decided to outline OpenMath unit converter system specification and its implementation in one section. These two aspects of the project are naturally connected. Therefore it is natural to describe the specifications (that is how the things should work) and implementation (how to actually make the things work) together within one context. This will include discussion of decisions on problems I faced as well as explanation how I came to the decisions (see, for example, section on user interface 3.3.3 on page 27).

3.3.1 Startup

As mentioned in System Requirements (section 3.1 on page 17), the only information required for the system to make unit conversions is to locate CD files and their corresponding STS files. Although it would be reasonable to store CDs and corresponding STS files in the same directory, this is clearly not the choice that OpenMath community has made. CDs and STSs are stored in different directories at the OpenMath site, so we decided keep a local file (*locations.cfg*) in the program script directory that contains separate locations for CD and STS directories.

The interface script is invoked in two cases: (i) to generate the input form containing unit and prefix information, and (ii) to start the conversion and report the result. In both cases the information on CD directories is read from *locations.cfg*. The directory listing should contain the minimum set of files in order to have at least one unit system available for conversion. This must be checked upon script invocation. The base unit system is SI system that is *metric*. This means that the following files are an absolute minimum for the converter to start up without errors:

- `units_metric1.ocd`
- `units_metric1.sts`

There is no way to determine which CDs belong to unit definitions because there is no definition file that lists these CDs. If we were to look for those CDs which contain "units_" at the beginning of the filename then this would include CDs like *unit_ops1*, *units_siprefix* & *units_sts*, none of which represent unit definitions (although they are used by unit definition CDs). One of the solutions would be to make a list of CDs which represented unit definitions by getting a list of CD filenames which start with "units_" and then exclude the list those CDs that are known not to define unit systems. However, this is a very inefficient way of solving the problem since the system will not be robust anymore: the CD list is constantly evolving and can be is extended with non unit defining CDs with filenames starting with "unit_".

There is another piece of information which might help determining what files are used for defining units: STS signature files attached to unit definition CDs all reference *dimensions1* CD. We can address this problem in the following way:

1. Get a list of CDs that have corresponding STS files. To filter them out from the list all names which don't have "unit_" prefix.
2. Check that the minimum pre-requisite files are present in the resulting list (the minimum files were listed earlier in this section).
   If they are not present, report an error and stop the program.

3. Read all STS files, extract useful information from them and put this information into two main hashes (associative arrays)

All STS files that start with "units_" are of the extremely similar structure:

```
<CDSignatures type="sts" cd="orig_cd_name">
<Signature name="signame" >
<OMOBJ><OMA>
    <OMS name="type" cd="cd_of_type"/>
    ...extra OMS & OMV tags possible here...
</OMA></OMOBJ>
</Signature>
...more signatures here...
</CDSignatures>
```

The important information is contained in *Signature* and *OMS* tags. It is this information that will go into two main hashes. How we process STS files and what we put into these hashes is described in section 3.3.2 on the next page.

Note that for CDs which define units, *cd* attribute in *OMS* tag has a value of "dimensions1" (even in *units_time1*!). We can determine which CDs contain unit definitions using this information. In fact, having this information means that we don't need to find out which CDs contain unit definitions. Required CDs will be loaded automatically when needed (described in section 3.3.5 on page 36) and the rest of CDs that define units will be loaded as they needed.

Script structure for this part of the system is :

```
invoking script ->
    Startup.pm
        sub:  ParseSts
        sub:  GetFileList
        sub:  Traverse
```

Invoking script is the script that is called from the browser. This can be a script that is called to display the interface or the code to carry out the conversions. The structure of the invoking script itself will be discussed in the appropriate sections (see User interface, section 3.3.3 on page 27 and Input parsing, section 3.3.4 on page 35). *Startup.pm* module will contain subroutines that are responsible for parsing STS files. *ParseSts* subroutine is called without parameters from the invoking script but returns either:

- references to two main hashes; or
- a scalar holding error information; or

- undef, which means that there was an error when trying execute *ParseSts*;

Path to config file (*locations.cfg,* as mentioned before) is defined inside *ParseSts*. The format of *locations.cfg* is the following:

```
cd::/path/to/cd
sts::/path/to/sts
```

ParseSts subroutine has the instruction to deal with extraction of directory paths from the *locations.cfg* using the stated format (use *split* function). If it is not able read the config file or there is not enough data in config file then it passes an error message to the invoking script and stop the program. Extracted paths to CD and STS directories are passed as parameters to *GetFileList* subroutine. The algorithm for *GetFileList* is the following:

1. Return an error message if the input parameters are incorrect (for example, when one of the parameters is missing);
2. Get a list of CDs, STS files for which exist. If there are problems in reading directory lists, return an error message;
3. Filter out all names which don't have "unit_" at the beginning;
4. Check that the minimum pre-requisite files are present in the resulting list (the minimum files were listed earlier in this section). If they are missing, issue an error message;
5. Return the reference to an array of filenames for STS files that we might find useful.

If *GetFileList* sub returned an error, or didn't return anything, then the startup script issues an error message and stops the program. If *GetFileList* subroutine receives a valid reference for an array, then *Traverse* script is called for each entry in this array to parse each STS file. The algorithm for *Traverse* subroutine is described in the next section.

3.3.2 STS file processing

As it was mentioned in the previous section, *Traverse* subroutine from *Startup.pm* deals with parsing of STS files. *Traverse* subroutine is called for each entry from the array that is returned by *GetFileList* function. To be more precise, the new object of the XML::DOM::Parser class is created. Then, for each filename from the returned array we use object's *parsefile* method to parse the document. This creates a reference to the document object that is finally passed to the *Traverse* subroutine. If at least one filename entry can not be parsed (e.g. the file can not

be opened or file contains wrong data) the *ParseSts* subroutine is immediately stopped and error message is passed to the invoking script.

It is useful to note that STS and CD files are in valid XML file format with one minor exception: CDs and STSs don't have XML declaration tags. This is just fine since XML parser does not complain if this tag is omitted. If XML parser would complain then the problem could be solved by loading each file into an array and adding an XML declaration tag at the beginning of this array and then using *parse* method instead of *parsefile*.

Let's now discuss what happens inside the *Traverse* subroutine. *Traverse* is a recursive function. It receives a reference to the current node as one of the parameters, gets a list of children for the passed node, processes the current node and calls itself again for each child node passing a reference to the selected child node. Of course, if *Traverse* receives a null reference then it immediately stops the current instance of itself. The case when a null reference is passed to the function usually happens when the function is called without passing a reference to it.

There are other parameters that *Traverse* subroutine is looking for. However, they are optional and are used for checking that the structure of STS signatures is correct. These parameters are: (i) name of the father element node, and (ii) two attributes that have been found. It is clear that these parameters are *null* when the *Traverse* function is called for the first time. We will talk more about this aspect later on page 26 when discussing how we process particular element nodes.

It is important to note that the major objective of parsing STS files is to extract maximum amount of information. This means that we ignore any inconsistencies within STS structure that might occur if some parts of STS do not follow DTD specs. In this case it is not necessary to issue an error message.

Before calling itself and passing a reference to a child node, *Traverse* function must check what kind of node is the current node. We are only interested in two types of nodes: DOCUMENT_NODE and ELEMENT_NODE. If the current node is not of the two types then the *Traverse* function returns without calling itself again. If the current node is a root node (DOCUMENT_NODE) then we just call the Traverse function again and pass it a reference to the first child of document node. If the current node is an ELEMENT_NODE the first step would be to get the name of the node and extract all attribute data from it. Several checks are done to look for particular names of the element node and some extra processing is carried out if the needed element name is found. This processing includes the following stages:

1. If the name of the element node is *CDSignatures* **and** attribute *"type"* has a value *"sts"* (remember that we extracted attribute data at the beginning of the function) **and** attribute *"cd"* has some value assigned to it then:
   get a list of children for the current node and for each child call the function again passing as a reference to the child node as a *first* parameter, passing

the name of the current element node (i.e. *CDSignatures*, this would become a father node in the called function) as a *second* parameter, and passing the value from the *"cd"* attribute as a *third* parameter.

2. If the element name is *Signature* **AND** the father element name is *CDSignatures* (remember that father name is passed as one of the parameters) **AND** there is an attribute for this node called *"name"* **AND** the first attribute passed exists then:
   get a list of children for the current node and for each child call the function again passing a reference to the child node as a *first* parameter, name of the current element node (i.e. *Signature*) as a *second* parameter, the value from the first attribute that was passed as a *third* parameter and the value of *"name"* attribute for this node as a *fourth* parameter.

3. If the element name is *OMOBJ* **AND** the father element is *Signature* **AND** two parameters are defined then:
   call Traverse subroutine for each child node of the current node, passing a reference to each child node, a father name (i.e. *OMOBJ*) and two parameters.

4. If the element name is OMS and father element name is OMOBJ **AND** two parameters are defined **AND** attribute data for the current node contains values for *"name"* & *"cd"* attributes then:
   record the data. Specifically, if the second passed parameter value contains "unit_" at the beginning of the string, then the data is useful and is put into two main hashes. In the first main hash named *sysData* we put a unit name (extracted from the second passed parameter) as a key and CD file name which defines this unit (first parameter that was passed) as a value for that key. In the second main hash named *unitData* we use again the name for the unit as the key and the value from the attribute *"name"* of the current node as the value to that key.

This concludes the STS file processing. Once all STS files are processed, two main hashes are created that contain enough information to produce user interface, parse data submitted by the user, check legality of conversions, etc.

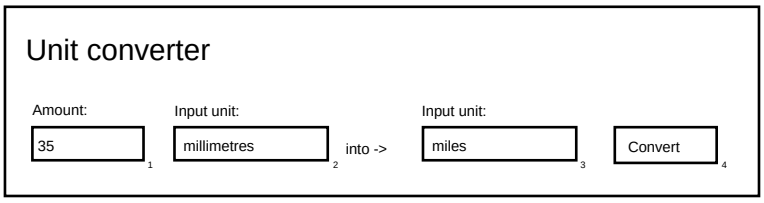
3.3.3 User interface

User interface design

Because Perl is used to develop the unit converter, the best interface can be created using HTML and controlled using CGI and a user agent (web browser). Before writing specifications for the User Interface, it is a good idea to discuss the possibilities that existed and why the particular user interface has been chosen.

Althouth there exist several web interface, we decided to use the simplest interface style available which is shown in Figure 3.1

Figure 3.1: Simple interface style



The interface we have chosen contains the minimum of elements with only three text input fields, see Fig. 3.1: field 1 for entering quantity of the unit to be converted, field 2 for specifying the original unit and field 3 for output unit. To carry out the conversion, "Convert" button is pressed (item 4). At first glance, this looks like a great interface (at least from programmers point of view) because it takes minimal amount of space, the program accepts minimal amount of parameters and you just type the units in the boxes instead searching them. In practice, this is a very ambiguous approach since there are so many ways to write each single unit (e.g. kilometre, kilometer, kilometres, km, etc) and it is not easily possible to make a program that will be able to deal with every variation of the every unit. Therefore, user must know exact spelling of each unit. This violates one of the system requirements outlined in section 3.1 on page 18, i.e. the system must "have a user-friendly interface and be intuitive to use". This interface is not intuitive and no one wishes to memorize the syntax to use the unit converter.

The second two versions of unit converter interface (Figures 3.2 and 3.3) are very intuitive to use: you just enter the amount in input box (item 1 in Figures 3.2 and 3.3) and make choices from listboxes (items 2, 3, 4 & 5 in Figures 3.2 and 3.3). The interface is divided into several parts (pages), each page being responsible for a separate measure, e.g. separate pages for length, pressure, area, etc. To select a specific page that allows you to do required conversion user has to click corresponding menu link (item 8 in Figures 3.2 and 3.3) at the top of the page. Under the menu there is also a headline that displays the type of currently displayed measure (item 7). The default page displays conversion page for the most frequently used measure (unit) which is certainly the measure of lengths.

The only difference between interfaces **a** (Figure 3.2) and **b** (Figure 3.3) is that the units are presented in menu boxes (items 2, 3, 4 & 5 in Figure 3.3) instead of listboxes (items 2, 3, 4 & 5 in Figure 3.2). Using menus instead of listboxes makes options easier to see and so they are more intuitive to use. Therefore, we would use interface **b** if there were no other alternatives. Both of these interfaces have one disadvantage: if you need another type of conversion different from that is currently being selected, you have to click on one of the menu links

Figure 3.2: User friendly interface, version **a**

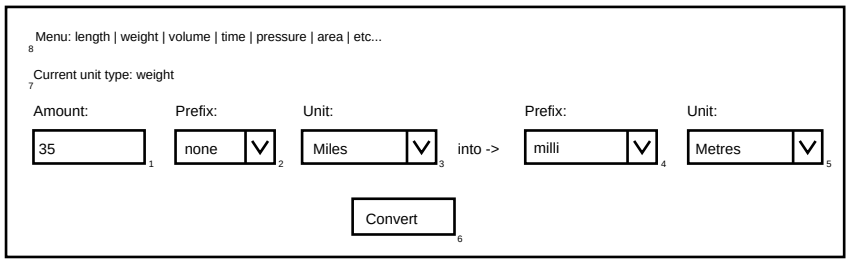


Figure 3.3: User friendly interface, version **b**

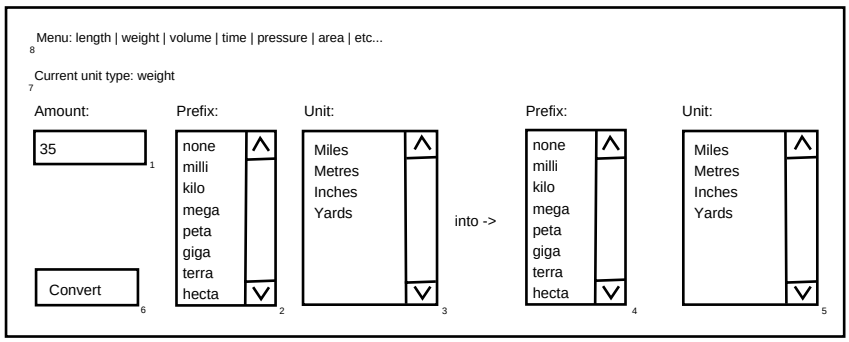
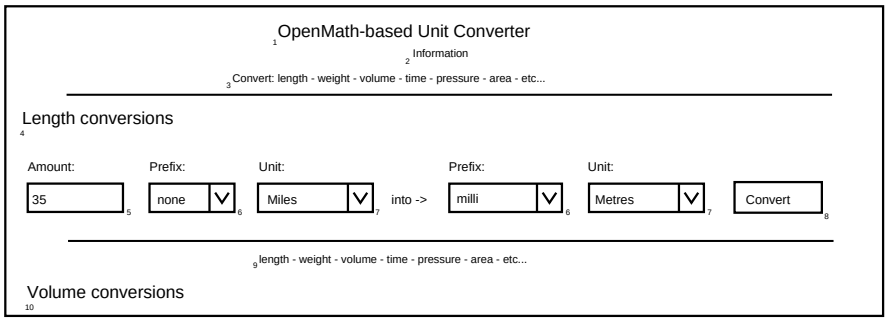


Figure 3.4: Section of the user friendly interface, the preferred version



at the top of the interface which will redirect you to a new page containing the interface for the desired measure conversion. The interfaces **a** and **b** are very much like the interface of freely available Windows converter reviewed in existing products section 2.8 on page 13. Having different measures on different pages is fine because the conversions are done offline, and there is no need for an additional data transfer in order to display an interface for different measure. As mentioned above, this is not a good feature in case of on-line converter.

The next interface shown in Figure 3.4 does not have problem just mentioned above: all measure conversions can be done from one page. There are links to the measure types at the top of the page. But rather than linking to other pages, these links point at different sections of the single page. All measure conversions are invoked using these links. Measure conversions are split up into appropriate sections. Each section has a title (e.g. Length conversion, see item 4 in Figure 3.4), input box for entering the amount of unit to be converted (item 5) and appropriate listboxes (items 6 & 7) to adjust the conversions (input/output units and input/output prefixes). "Convert" button (item 8) is also provided for each section. Sections are divided by the same links menu (for getting the necessary measure types) listed at the top of the interface. The interface is almost the same as interface in Figure 3.2, the only difference is that all measure conversions are listed on a single page and the top menu now points at different sections of this page.

We decided to use the latter interface shown in in Figure 3.4 since it is most user-friendly and easier to implement among all three interfaces considered. Below we discuss particular details of the selected user interface.

There is an "I*nformation*" link (item 2) below the heading "*OpenMath-based Unit Converter*" (item 1) which leads to the information page. This page contains basic information about unit converter including the version number and menu with list of available measures for conversion. The latter are linked back to the main interface page where all the conversion forms are displayed. The menu is exactly the same as that on the main interface page. Below version number it lists all the CDs that the converter has found. For each CD there is a list

of objects that a particular CD defines, this section is entitled as "Content Dictionaries available". The next section below section describing CDs (and objects inside them) is "About & Credits" section that gives the name of the author and a list of credentials to people contributed to the converter.

We return back to the main interface page and describe the next item after the "Information" link. This is the menu that gives a list of available measures available in the system for conversion. This list is the list of links of type:

```
<a href=''show.pl#measure''>measure</a>
```

They are arranged in alphabetical order for easy navigation between the measures. After the first menu, we list all sections for converting measures. Each section contains a title (items 4 and 10) that displays what kind of conversion each particular section does. The conversion form is given after the section title. This includes an amount box (item 5) that has a default value of 1 already present upon converter invocation, and listboxes for input/output prefixes and for input/output units (items 6 & 7).

Prefix listboxes. It is impossible to specify exactly which prefixes are available for which measures. Therefore, the prefix listboxes are the same for all input/output prefixes in all measure conversion sections. Prefix listbox contains entries sorted in an alphabetical order (so that it is easier for user to choose the required prefix) and a value of "none" is at the beginning of the list so that conversions without prefix values can be done.

Unit listboxes: They contain only units that are relevant to the particular conversion section. Before listing the available units, some simplification of listbox values were performed for more neat look, such as

- '_per_' replaced by '/'
- '_sqrd' replaced by '^2'
- '_' replaced by ' '
- make first letter upper case.

Unit listboxes are also sorted. Finally, the navigation menu is displayed at the end of the conversion form. As mentioned earlier, this menu is used to divide different measure conversion sections (item 9).

One might argue that conversion sections don't have to be displayed for the measures that have only one unit because there is nothing to convert between. For example, Watt is the only measure for power for both metric and imperial systems. However, a user that does not know numerical definitions of prefixes might want to convert milliwatts into megawatts. One might argue that everyone knows numerical definitions of these prefixes and can do conversions without

using converter. However, this is not completely true since there are some rarely used prefixes such "yocto" or "pico" that many people simply don't know. In addition, one might want to convert from one prefix to another prefix, e.g. from kilometres to millimetres. Therefore it has been decided to leave the conversion facilities even for the units that don't have any other equivalents.

User interface implementation

After developing clear understanding how the interface scripts will generate the listboxes and menus we identified the following steps in the converter flowchart:

1. to get a measure list from the hash where measures are the values of keys in *unitData* hash;
2. to get a CD list from the hash where CDs are values of keys in *sysData* hash;
3. to get a list of units for a particular measure, where a measure is equal to some values in *unitData* hash;
4. to get a list of units for a particular CD, where a CD is equal to some values in *sysData* hash;
5. to get a list of prefixes, where prefixes are some keys in *unitData* hash.

Obviously, the items 1 & 2 and 3 & 4 are performing exactly the same operations but on the different hashes, so it was decided that some portions of the code will be re-used. Small module *ExtractData.pm* deals with these items, that includes subroutine *GetUniqueValues* to deal with items 1 & 2, subroutine *GetKeysFromValue* to deal with items 3 & 4 and subroutine *GetPrefixes* to deal with item 5. Let's discuss them in more detail.

GetUniqueValues
It receives a reference to a hash as a parameter and returns a sorted list of unique values found in this hash. Note that this is not a list of unique keys. Therefore, for each key in the hash we put its value into a new hash as a key and assign a value of 1, in order to form a list of the unique values. This way non-unique list of values from the original hash will appear as a unique list of keys in the new hash. Then, we just return a sorted list of keys from the new hash.

GetKeysFromValue
It receives a reference to a hash and a value that is to be searched for in this hash. Returns a list of keys found that have a value passed to this subroutine. In order to get a list of keys for the value we do the following for each key: if its value is equal to the one passed to the subroutine the key is put into a new hash as a key. After this operation, a sorted list of keys from the new hash is returned. We could just push new keys as values into a new array instead of a

new hash because keys will not have duplicates as they are also keys from the hash. However, use of a new has makes the the code more robust.

GetPrefixes
It receives a reference to a *unitData* hash and returns a list of prefixes. In order to get a list of prefixes for each key in the *unitData* hash, we check if its value is equal to "unit_prefix". If it is, then add the current key from *unitData* hash as the key to the new hash. Finally return a sorted list of keys from the new hash.

There are two scripts that are responsible for displaying the user interface: *show.pl* and *info.pl*. *Show.pl* deals with the main page generating forms for converting different measures and *info.pl* displays the "Information" page. Designs and layouts for both of these have been described in the previous section.

Both *show.pl* and *info.pl* display a heading "OpenMath-based unit converter" when executed from the browser. They then try to run external ParseSts function (described earlier in section 3.3.2 on page 25) and to get references back to two main hashes. Then, both scripts check if both returned variables are references. If they are, then the steps to produce an appropriate interface are taken, But if they are not, we check to see that first returned variable exists and is not a reference. If this is true then we print an error contained in the first variable. Otherwise, we just stop the program and report that an unknown error has occurred as *ParseSts* did not return anything. After this check, the main code for displaying the main interface and information interface branches onto to parts:

show.pl
If two variables are references then show.pl does an extra check to see that one of the main hashes (it doesn't matter which one as they will have the same keys) contains "units_metric1" value for at least one of the keys. If this is not the case then the script should report an error stating that there is not enough information to continue. The situation like this should never happen in normal circumstances as *ParseSts* performs some checks to see that there is a "units_metric1" STS and CD present. However, what it does not do is to check that there is at least one definition inside that STS file. As mentioned above, this normally will not happen but it is still useful to have such a check in the system. This makes the code much more stable and robust. This check is not necessary to be done in info.pl as it is only responsible for displaying information about available CDs and author/credits info.

If the test has passed successfully (i.e. at least one definition of metric unit was found) then the first thing to do is get a measure list by calling a *GetUniqueValues* function from *ExtractData.pm* and passing it a reference to *unitData* hash. Measure list has to be processed because it includes non-measure values, such as "units_sts" for prefix entries. We also have to generate menu entries using this list. Thus, for each entry in the raw measure list we check to see if it starts with "unit". If not, we add this entry to a new measure list and also add this entry to the list of menu entries (wrapping link HTML code around each entry before adding it to the menu list).

Next, get a prefix list by calling *GetPrefixes* function from the same module and passing it a reference to *unitData* hash. According to the user interface (UI) design specifications the prefix list is the same for all measures so we generate prefix listbox HTML code straight away by concatenating all values from prefix list along with HTML wrappers and storing this as a single variable.

These steps were preparations for displaying the main UI. Next, we display the first menu by concatenating values from the menu list. After this, measure conversion sections is displayed. For each entry in the measure list we do:

1. get a unit list by using *GetKeysFromValue* (from *ExtractData.pm*) and passing it: a reference to the *unitData* hash and a current measure value;
2. prepare unit listbox by replacing all substrings in each entry (as described in section 3.3.3 on page 27), uppercasing the first letter, wrapping each unit with HTML listbox code and concatenating all unit listbox values into a single listbox;
3. print the title of the current section (uppercased first letter of the measure name followed by a word "conversions", e.g. "Length conversions");
4. display the conversion form by printing a table with amount input field, "input prefix" listbox (using the prefix listbox code prepared earlier), "input unit" listbox (point 2 above creates it), "output prefix" listbox (same as "input prefix" listbox), "output unit" listbox (same as "input unit" listbox) and a submit button to trigger the conversion process.
5. finally, display the menu underneath each measure section by concatenating entries from the menu list.

info.pl
After performing the checks for valid references to two main hashes as described earlier, *info.pl* gets a list of available measures and checks each measure that contain "unit" at the beginning. If it doesn't contain this forbidden substring then the measure is wrapped up in a link HTML code and is inserted into a menu list. The script then gets a list of available CDs by calling *GetUniqueValues* and passing it a reference to *sysData* hash. The actual interface is then displayed as follows:

1. print a menu using the menu list generated earlier;
2. print a heading to define the section listing of all defined objects: "Content Dictionaries available";
3. for each entry in the CD list, print the current CD name and appropriate objects defined in the current CD (by using *GetKeysFromValue* and passing a reference to *sysData* hash and a CD name as arguments);
4. after all CD names and appropriate objects have been listed, the script prints author and credits info.

3.3.4 Parsing user input

process.pl script is responsible for getting user input, executing an external function to do the conversion and displaying the result of this conversion to the user.

To get the data submitted by the form, the program should use generic CGI module that comes with standard Perl installation. "Amount", "input prefix", "input unit", "output prefix" and "output unit" are all stored as separate variables. There is also a special *CheckData* subroutine in *process.pl* that validates all user data. Thus, after storing all our variables, we call *CheckData* subroutine with all user input variables as parameters. *Then, CheckData* function checks that:

1. amount entered is a positive number,
2. prefixes exist and are in fact prefixes,
3. units exist and are in fact units,
4. type of "input unit" is the same as type of the "output unit".

If any of these checks didn't pass then the *CheckData* subroutine prints an error message and returns *null*. If all the data are valid, the subroutine returns a value of 1. Therefore, when calling *CheckData* from *process.pl* we must check the return value of the function. If the function didn't return anything then we stop the program (no need to report an error as this has already been done by the CheckData sub). Note that all checks are separated into a subroutine to avoid unnecessary nesting. The subroutine is defined inside *process.pl*.

If *CheckData* returned without errors, we take several steps to produce the conversion:

1. get the numerical values of "input/output prefixes";
2. produce global coefficient by multiplying the amount by "input prefix" value and then dividing by "output prefix" value;
3. get the value resulting from conversion;
4. get the final result by multiplying global coefficient and the value resulting from conversion;
5. display the final result to the user.

Notice that prefixes and units are worked on separately and combined together only when producing the final result. Let's now discuss steps mentioned above in detail:

3.3.5 Conversion system

3.3.5.1 Common functions

Before implementing the main converter code, we can identify further several parts of processing that can be generalised for the system and thus allow more code re-using:

1. We need a function that can parse documents using XML::DOM::Parser module. This could be used by the prefix converter and especially by the unit converter when doing recursive processing (and so several CDs might need to be parsed),
2. We need a function that can find a reference to the specific object definition given an object name and a reference to the document object of the CD that contains a definition of this object. This function will also report what the selected unit is defined in terms of,
3. We need a function that will emulate standard arithmetic operations.

Let's consider these in detail now:

Parsing Content Dictionaries

The purpose of *ParseCd* subroutine is to try and open CD, parse it with XML parser and return a reference to the document object for that CD given a CD name.

ParseCd should receive a name of the CD file. If this subroutine was called without parameters then it should produce an error report and stop. The function then opens a configuration file (path to this file should never change and so is embedded in the subroutine as a variable). Paths to CD and STS directories are then extracted from the opened configuration file. The function should produce an appropriate error report and stop if the configuration file could not be opened or the paths to either STS or CD directories are not found. Using the directory path for CD files, an appropriate CD should be parsed using the described method for parsing STS files, see section 3.3.2 on page 25 (path given to *parsefile* method of XML::DOM::Parser consists of CD directory path extracted from configuration file and a CD name that was passed to this subroutine). If the XML parser could not process a file, the function should catch an error report generated by the parser, display it to the user and stop. Reference to the parsed CD document object is then returned. Note that this reference should also be inserted into the *cdDocuments* hash that stores CD names as keys and references to those CD document objects as values to the keys. This will eliminate the need to parse the same CD more than once, because next time we can get a reference to a parsed version of this CD from the *cdDocuments* hash.

Finding object definitions

FindObject subroutine is needed to find the reference to the FMP part of the definition given a reference to the parsed CD document object and a name of the object, definition for which is required. In fact, the returned reference should point not to the *FMP* tag but rather to the first *OMA* tag as this will reduce the amount of work that needs to be done by the code that calls this subroutine when processing objects.
FindObject subroutine should also return a list of objects that are used to define the selected object.

So, as soon as the input is received by the *FindObject* function, we check that both input arguments received (reference to the CD document object and an object name) are valid and get a list of references where *Name* tag is found in a CD. The function then, for each entry in the list of *Name* tag references, checks whether the first child (i.e. data within <Name> and </Name> tags) is the same as the selected object name. If none of the nodes in the list meet the described criteria, then this is an error as the object is defined by one of the STS files but is not defined in the appropriate CD. Thus, in such case function displays an error report and stops.
If the name inside one of the *Name* tags is equal to the name of the object that we are looking for, then the function has found the needed location of the object definition. But this is not entirely correct location as it points to the *Name* tag. What is required is the location of the first *FMP* tag, which is one of the siblings of the found *Name* tag. The best method to browse to this tag is to:

1. get the parent element of the *Name* tag. If the parent element name is not *CDDefinition* then display an error reporting that the definition of the selected object is corrupt and stop the subroutine,
2. find all sub-elements of *CDDefinition* that are named *FMP*. If there are none found then return a value of 1 and stop. Depending on a CD being parsed, this might mean different things: corrupt definition inside prefix CD, for "units_metric1" this will mean that a unit is a base SI unit (unless incorrectly defined) and for the rest of unit definition CDs this will also mean that the definition of a unit is corrupt. What action to take in this case is the responsibility of the code that called this function,
3. find all sub-elements of the found *FMP* tag that have the name *OMOBJ*. If none found then this an error, the definition of an object must be corrupt so display an appropriate error report and stop,
4. find all sub-elements of the *OMOBJ* tag that are named *OMA*. If none found, display an error stating that definition is corrupt and stop. Else, the first reference to an *OMA* tag that was found is the reference we are looking for. Check that parent of this reference is an *OMOBJ* tag and report an error if not. This reference will be returned later by a function as one of the parameters.

Further checks need to be done to ensure the validity of a definition. Because there are so many possible ways to define objects, function can not check every-

thing, but there are some rules that all definitions must obey:

1. first child of the first *OMA* tag should be an *OMS* tag which will relate next two children (i.e. and *OMS* tag with *cd* attribute equal to "relation1" or "set1" and *name* attribute existing),
2. there are either one or three children for each element node.

After further validation, function should now start extracting all objects that are used to define the selected object. This is done by getting a list of all *OMS* tags that are sub-elements of the third child of the initial *OMA* tag (the one which is a child of *OMOBJ* tag) including processing the third sub-element itself. For each *OMS* tag that was found we then check that *cd* attribute value starts with "unit_". If it does, add a value of the *name* attribute to the array that contains all names of the objects that were used to define the selected object. After processing all *OMS* tags, function should return a reference to the beginning of the object definition (i.e. reference to the first *OMA* tag) and an array of objects that were found to be used for the selected object definition.

Emulating standard operations

The structure of OpenMath Content Dictionaries is such that objects are defined in terms of some other objects, which in turn, are also defined in terms of other objects. This recursive process goes all the way down to the core set of Content Dictionaries ("Core" section of the OpenMath website). It is pointless to try and produce a unit converter that will be able to understand all definitions all the way down to the "Core" set. After all, the objective of this project is to demonstrate usefulness of the proposed extension of unit CDs, so such precise and time-consuming approach is completely unnecessary. Thus, at some level we will need a function that emulates object definitions for some CDs.

As we know from the section on background information (page 2.7 on page 10), operations defined in "arith1" CD are used to attach units to numbers and other units. This would be a very sensible point to start the CD emulation.
The purpose of *arith1* subroutine is to emulate the operations that are defined within "arith1" CD. *arith1* should receive the operation name and two arguments to perform an operation on. It should then check whether there are three arguments passed to it and if they are not null. In case of an error, the function should display a message to user and stop the program.
It would be useful to have definitions of reverse operations emulated as well because this will allow us to perform reverse conversions. This could be done by passing to *arith1* an operation name with a minus sign in front of the operation. Thus, *arith1* should check if first symbol of operation name is equal to "-". If this is true, *arith1* should take steps to redefine the reverse operation, e.g. if operation name is "-plus" then the function should exchange two arguments (first argument becomes second argument, second becomes first) and operation name should be changed to "minus". After checks for reverse operations, the function should perform the selected operation and return the result value.

3.3.5.2 Getting the numerical values of "input/output prefixes"

When doing conversions with prefixes, we need to convert prefixed values as well but prefixes are just factors (input prefix should be multiplied by the amount before amount is being passed to the unit converter code, result of conversion is divided by the output prefix value). Thus, prefixes need to be processed separately from unit conversions. Function *GetPrefixValue* does exactly that. It receives prefix name as well as the reference to the *sysData* hash (this hash stores the names of CDs which hold a definition for the particular object). This function should then output the numerical value of the selected prefix.
As soon as *GetPrefixValue* receives parameters, it should check that prefix name is not an empty string and that the hash reference is a real reference. It should then parse the CD that defines the selected prefix (we can find out the name of the CD that stores the needed definition by accessing *sysData* hash and we can parse the selected CD using the function *ParseCd* that was described earlier in section 3.3.5.1 on page 36). The function should then try to find the location of the selected prefix definition within the parsed document (using the *GetObject* subroutine described in section 3.3.5.1 on page 36). If definition was not found, *GetPrefixValue* should display an error report and exit.

The structure of all prefixes is the same. Moreover, the main structure is not supposed to be changed so it is decided to implement a non-recursive extraction of values from prefix definitions. This way we can ensure that only fully prefix-compliant structures are used.
So, using the the output from *GetObject* we check that the element at the found location is an *OMA* tag. If not, the function should inform user that the definition is invalid and exit. The subroutine should then get a list of all children (element nodes only) of the found *OMA* tag and check that the first child is *OMS* tag, second child is *OMA* tag and third child is an *OMV* tag. If this condition is not met, function reports that the definition is corrupt and exits. Next, the function should check that attribute *name* has a value of "times" and *cd* attribute has a value of "arith1". If this condition is not met, function should display an error and exit. Next check is preformed on the third child. We get the value for the *name* attribute of this node and ensure that it is equal to "unit". If not, report an error and exit. Finally, we get a list of children of the current second child (which passed all previous checks so must be an *OMA* tag). Using the new list of children we need to check that the first child is an *OMS* tag, second and third are *OMI* or *OMF* tags. If this check was not passed, function reports "corruption" error and exits. If the condition was met, the function calls *arith1* subroutine passing it an operation and two arguments. Operation name is extracted from the *name* attribute of the first child (*OMS* tag), first argument is extracted from the second child and second argument is extracted from third child. Before extracting operation name from the *OMS* tag, the function should check that *cd* attribute for this tag is equal to "arith1". After *arith1* was called, we check to see what it returned. If it did not return anything, an error must have happened inside *arith1* so *GetPrefixValue* should exit. No need to report an error as this must have been done inside *arith1*. If *arith1* did return a value then this must be a numerical value of the prefix so we also return this value from *GetPrefixValue* subroutine.

3.3.5.3 Unit converter

So far we have been discussing various "wrappers" around the unit converter that will either provide some interface functionality or some helpful functions that could be used by the unit converter. The main and most complicated part of the code is the "engine" of the system which is a unit converter itself. There is a lot required from this part of the code, the algorithms involved in this section of the system are complicated and involve considerable amount of recursion.

Before thinking about how to create conversion algorithms, we need to classify all possible conversions. This would assist in creating the best algorithms for conversions. If we ignore prefix data (we dealt with it in the previous section), then the following classes exist:

1. non-base to base primitive conversions (e.g. degrees Fahrenheit into degrees Celsius)
2. non-base to base non-primitive conversions (e.g. miles into metres)
3. base to non-base primitive conversions (e.g. metres into feet)
4. base to non-base non-primitive conversions (e.g. metres into miles)
5. non-base to non-base primitive conversions (e.g. miles into feet)
6. non-base to non-base non-primitive conversions (e.g. feet into U.S. survey feet)
7. composite conversions (e.g. metres per second into miles per hour)

where primitive conversion is when one of the units is expressed in terms of the other one directly.

Clearly, class 1 is easy to calculate - you just take a CD definition of the input unit (non-base in this case) and perform all operations contained in the CD substituting amount at the position where *OMS* tag for the output unit is found.
Class 2 is a little more complicated - you look at what input unit is defined in terms of. You then convert from input unit into the unit that that is used to define an input. The same step is repeated again for the new unit and so on until we get down to the bottom, i.e. to the base unit. Basically, to get this done we use the algorithm from class 1 in a recursive way.
Class 3 is also easy to calculate, it is similar to class 1 but operations are done in the inversed way when compared to that of the output unit definition. We perform these inversed operations on an amount.
Class 4 is more complicated. It is similar to class 3 but recursion is used to get from the output unit to the base, passing an amount right to the bottom. Once recursion procedure gets to the bottom, the first inverse conversion is done using the amount that was passed. The value returned from conversion is the new amount that is used to perform inverse conversion one step up and so on until the recursion gets back to the top.

Class 5 is really simple and is exactly the same as either class 1 or class 3 (depending on whether the input is defined in terms of output or output is defined in terms of the input).
Class 6 is the most complicated conversion that is possible. The method for carrying out this type of conversion can be thought of as moving from right to left. First step converts input into base using forward recursion and second step uses inverse recursion to go from base unit to the non-base output unit.
Class 7 is simple and can be thought of as applying class 6 to separate components of the composite unit and then assembling them together.

There will be a *UnitConverter.pm* module that would deal with all conversions. This module will have the following structure:

```
invoking script ->
    UnitConverter.pm
        sub:  ConvertUnits
        sub:  ForwProcessOma
        sub:  BackwProcessOma
        sub:  CompositeProcessOma
        sub:  GetPrefixValue
        sub:  ParseCd
        sub:  FindObject
        sub:  arith1
```

As well as subroutines for converting units, this module should have subroutines that deal with prefix processing (*GetPrefixValue*) and assist in conversion process (*ParseCd*, *FindObject*, *arith1*) all of which have already been described.

ConvertUnits subroutine is responsible for initialising the conversion process. It is initially called from the process.pl script which does the user input parsing and starts the conversion. *ConvertUnits* should be able to perform a classification described earlier in this section and call other subroutines depending on the classification result. If the *ConvertUnits* detected that required conversion is of class 1, class 2 or class 6 then it should call *ForwProcessOma* function to do the conversion.
In case of class 1 conversion, *ForwProcessOma* is called with the following parameters: reference to the input unit definition, name of the output unit and amount.
In case of class 2, we find out what unit is used to define the input unit and call *ForwProcessOma* passing it a reference to input unit definition, name of the unit that is used to define input unit and amount. The output returned by *ForwProcessOma* is then recorded and we call ConvertUnits again but now passing the new amount (the value that was returned by the *ForwProcessOma* in the previous step), the new input unit name (this should be a name of the unit that was used to define the original input unit) and output unit name. This ensures that the recursion goes one step down at a time and will gradually reach the base SI unit.
In case of class 6, the same steps are taken as per class 2. The recursive step

of class 2 processing will ensure that when we reached the base input unit, the next time *ConvertUnits* is called, the conversion will be detected as class 3 or 4.
One of two cases of class 5 (case when input non-base unit is defined in terms of the output non-base unit directly, labelled as "class 5 part 1" in the code) should also be processed by *ForwProcessOma* using the method that was applied to class 1 conversion.
If the *ConvertUnits* detected that needed conversion is of class 3 or 4 *BackwProcessOma* is called passing it a reference to the output unit definition, input unit name and amount. The recursion for class 4 is initiated within the *BackwProcessOma* function.
In case of second part of class 6 (output is defined in terms of input, labelled as "class 5 part 2" in the code), the function should also call *BackwProcessOma* passing it the same parameters as per classes 3 and 4.
If *ConvertUnits* detected class 7, we need to call *CompositeProcessOma* passing it references to input and output object definitions and the amount.
Note that *ConvertUnits* subroutine should be able to deal with returned values of zero, negative and interval definitions.

Let's discuss the inside algorithms of the conversion subroutines mentioned above:

ForwProcessOma

This is a recursive function that parses through a definition tree (one level is processed per call) and computes the numerical values as it goes along. If it finds an *OMS* tag that references the output unit, the amount is substituted in this place and numerical operations are done as normal. The final amount is returned back to the calling function.

BackwProcessOma

This is also a recursive function that parses through the definition tree, but this time, we need to get a value that will then be used together with amount to perform an inverse operation. To get the amount for each node we do a trick by using the ForwProcessOma and passing it references to children nodes (only for second and third child as the first one is always an operation), some random unit name and arbitrary amount. If the current child tree only involves numerical objects then the value is returned by *ForwProcessOma*. If the current node does involve some unit definitions then there will be nothing returned back by *ForwProcessOma* as we asked it to look for a unit definition which has some random non-existent name.
Another recursive part of *BackwProcessOma* is initiated if the output unit is not defined directly in terms of input. In this case, before returning any results or performing any operations we call ConvertUnits subroutine again passing it the original amount, input unit name and the unit found in the current output unit definition as a new output.

CompositeProcessOma

This function just splits up the composed input and output definitions into components, then performs unit conversions between appropriate unit pairs using *ConvertUnits* and passing 1 as an amount. It then composes back values returned by *ConvertUnits* using operations found in the definition of the composed input unit and any numerical values found in the composed definition. If there are numerical values existing in the composed output unit definition then those values should be applied to the result as well using the reversed operations.

Let's now discuss the code that provides an output to the user.

process.pl is responsible for displaying conversion results to a user.
First of all, the script does the common checks, similar to the ones done in interface generator, that check that *sysData* and *unitData* hashes exist and that SI unit system (metric) is present. If these conditions are not met, the script generates an error report and exits. Next, script parses input as described in section 3.3.4 on page 35. After this, we call *GetPrefixValue* from the *UnitConverter.pm* to get the numerical values of input and output prefixes. We then get the new amount by multiplying original amount by the input prefix value. We then call *ConvertUnits* subroutine from *UnitConverter.pm* to produce the unit conversion and we pass it a new amount, input unit name, output unit name and references to *sysData* and *unitData* hashes. The value received back from *ConvertUnits* is divided by the output prefix value and the result is displayed to a user.

Note that conversion algorithms have not been discussed in as much detail as the other parts of the system for the following reasons:

- This helps the reader to concentrate on conversion algorithms rather than on technical details of implementation,
- The source code is provided for the converter code (the main converter algorithms) in appendix D on page 66 which can be used to find out implementation details.

3.3.5.4 Final product

The converter has been implemented according to specifications, testing report in the next chapter identifies how well the system was implemented. Only source code for main converter algorithms is provided in the printed version. Full source code set is available on the CD that comes with this dissertation.

Chapter 4

Testing

In course of implementing this project two types of testing have been performed: development testing and requirements testing. The development testing of the system aims at performing a set of measures that guarantee that the system is robust and bug free (although it is almost impossible to exclude any bugs even in commercial products). As it was mentioned in the Resources section 3.2.4 on page 21, Mantis bug tracking system was used to track all bugs that were found in both OpenMath data and the converter system during development testing. Mantis has also been used to monitor how bugs were resolved. The bug tracking system can be accessed online at: `http://www.bath.ac.uk/~ma1eo/mantis/` (login: guest, password: guest). A short summary of all bugs found is given in the appendix C on page 65. Special care was given to error processing discovered at the stage of development testing. All fault situations have been simulated to ensure that all checks for corrupt or invalid OpenMath data produce error messages and exit.

When testing parsing function, a custom form page was created that includes all the input fields as text fields. This enabled us to input values which were not possible to enter using the standard user interface. This was then used to simulate situations when some data got lost while being transferred from the browser to server and to check that input validation is functioning correctly.

The second type of testing, requirements testing, focuses on comparing the final version of the system against a set of system requirements (chapter 3.1 on page 16). Before completing requirements testing, the following tasks were performed:

1. To check that all CDs contain full and valid definitions. Bugs within CDs have been found while completing development testing and are noted in the bug tracking report (appendix C on page 65).
2. A custom dictionary was made for some new measure system so that we could test that new units and unit systems that might be added easily to

the converter in the future. (This is one of the requirements as formulated in the chapter 3). Our choice included U.S. units as a custom dictionary that has been created for testing.

We shall briefly discuss rules that were followed when creating the dictionary. New unit system was defined to represent the U.S. units, two files were created:

1. "units_us1.ocd" - a CD that defines U.S. units in terms of the SI ("units_metric1") units,
2. "units_us1.sts" - an STS file that describes the type of each unit defined in "units_us1.ocd".

What has to go into "units_us1.ocd"? The U.S. customary units (more commonly known in the U.S. as "English units") are the non-metric units of measurements that are presently used in the United States alongside the metric system of units [7]. Most of the measurements of this system are the same as measurements from the imperial (UK) system with the following exceptions:

1. There exists another foot definition which is a U.S. survey measure [1]. Imperial measure states that 1 foot is equal to 0.3048 metres exactly, while U.S. survey measure defines 39.37 U.S. survey feet as 12 metres exactly (one foot = 1200/3937 meter). This causes yards, miles and acres also be different (although the ratios between them are exactly the same as in the Imperial system). So "units_us1" must contain definitions for the following units:
   (a) *foot_us_survey* (1 U.S. Survey foot = 1200/3937 metres),
   (b) *yard_us_survey* (1 U.S. Survey yard = 3 U.S. Survey feet),
   (c) *mile_us_survey* (1 U.S. Survey mile = 5280 U.S. Survey feet),
   (d) *acre_us_survey* (1 U.S. Survey acre = 4840 square U.S. Survey yards),
   (e) *inches_us_survey* should be defined as well, but "units_imperial1" misses them out so we trust the creator of "units_imperial1" that we do not need to define inches *inches_us_survey* in "units_us1" for consistency.
2. Volume units for measuring liquids are different for the U.S. system (i.e. cubic foot, cubic inch and cubic yard are the same as in Imperial system but others are different). We ignore survey volumes because survey units are used for cartography only. This means in our case that U.S. pint must be defined in "units_us1" in terms of SI base unit, which is litres. The definition of U.S. pint states that 1 U.S. liquid pint is equal to 0.473176473 litres exactly. Note that unlike Metric or Imperial, U.S. system has separate types of units - one type for dry volume and another for liquid volume. Therefore, definition of dry pint also exists, one dry pint being equal to 0.5506104713575 litres exactly.
   It follows that the following new units have to be added to "units_us1":

(a) *pint_us_dry* (1 U.S. dry pint = 0.5506104713575 litres),

(b) *pint_us_liquid* (1 U.S. liquid pint = 0.473176473 litres).

Information that confirms the conversion facts stated above is available from the Wikipedia encyclopedia [7] and from the number of other official sources. "units_us1" CD and STS files have been created according to the notes above and are listed in appendix E on page 67. With this new measure system present, we performed full requirements testing. Results of this testing are listed in the table below.

Test	**Outcome**	**Comments**
Get a list of available unit systems from the configured location		
Does the system handle configuration files correctly?	success	
Does the system report if configuration file contains erroneous information?	success	reports an error if at least one path is missing
Does the system find the correct paths?	success	reports if directories can not be opened
Does the system detect CDs that represent units?	success	produces an error if SI CD ("units_metric1") is not found
Is it easy to customise the name of the SI system?	failed	you will have to go through several scripts to replace the name of the SI CD
Does the system check that SI CD has at least one definition?	success	produces an error if no units are found in SI CD
Does the system detect prefix CDs?	success	
Does the system detect all available signatures for objects found in CDs?	success	lists all detected objects on the information page
Does the system detect the type of object?	success	
Load CDs and all other necessary files needed for conversion		
Is the system able to load and parse the CDs and STSs required during the conversion process?	success	references to CDs that have previously been parsed are stored in a hash, thus saving time and memory when processing composed and recursive conversions
Work with any number of unit systems and easily add new unit systems		
Is it possible to add new units/prefixes?	success	added objects appear in the interface next time it is displayed

Test	Outcome	Comments
Is it easy to add new units/prefixes to the converter?	success	to add a new object, all that needs to be done is to update the relevant CD and STS files
Is it possible to add new unit systems to the converter?	success	objects defined in the new system files appear in the interface next time it is displayed
Is it easy to add new unit systems to the converter?	success	to add a new unit system, just drop CD and STS files for the new system in the directory which is referenced by the converter to get CDs and STSs
Do conversions work between already created systems and the newly created custom systems?	success	
Is there a limit of number of systems that can be added?	success	no limit on the number of systems that can be added, but a lot of systems will make interface more difficult to use; clearly an interface issue

Interpret CDs (parse the format and understand the data)

Is the system able to understand the CD structure?	success	
Is there any validation of CD data?	success	common rules are checked when processing CD data
Does the system understand composite units (like Acre)?	success	works on components of the unit separately and then joins together to finalise the conversion

Display units/prefixes available for conversion

Does the system list all units that have been found?	success	both main interface and information page list all available units
Are the units listed in type groups?	success	interface consists of measure sections, each section contains all units that are of that particular measure type
Are the units which have no other equivalents displayed as well?	success	"single" units are displayed as well so that prefix-only conversions can be done
Are all prefixes listed for each measure section?	success	prefix list is the same for all sections

Get user input and process it

Test	Outcome	Comments
Does the system get input from user?	success	
Is the input validated?	success	system checks if all required fields have been sent, checks the validity of the amount and existence of input/output prefixes and units
Does the system check if the selected conversion is valid?	success	before converting, the system will check that input and output units are of the same type
Does the system check if it possible to do the conversion?	success	if some definitions are incorrect and there are more than one base units, the system will report and error

Produce needed conversions

Is the system able to process input/output prefixes?	success	prefixes are generated before converting units themselves
Is the system able to do forward non-base to base primitive conversions?	success	e.g. degrees Fahrenheit into degrees Celsius
Is the system able to do forward non-base to base non-primitive conversions?	success	e.g. miles into metres
Is the system able to do backward base to non-base primitive conversions?	success	e.g. metres into feet
Is the system able to do backward base to non-base non-primitive conversions?	success	e.g. metres into miles
Is the system able to deal with non-base to non-base primitive conversions?	success	e.g. miles into feet
Is the system able to deal with non-base to non-base non-primitive conversions?	success	e.g. feet into U.S. survey feet
Is the system able to deal with composite conversions?	success	e.g. metres per second into miles per hour

Have a user-friendly interface and be intuitive to use

Is the user interface available?	success	web-based user interface is provided
Is it easy to get to know how to use the user interface?	success	it is clear for a new user how to use the interface; user manual is also provided, see appendix B on page 62

Test	Outcome	Comments
Is the user interface easy to use?	partial success	the interface is easy to use, but might become too chunky if many units/systems are added

Be easy to install and configure

Is the system easy to install by the unfamiliar user?	success	system is extremely easy to install; installation is described within the user manual that is given in appendix B on page 62
Is the system easy to configure?	partial success	paths to CD and STS directories can be configured by editing "locations.cfg" file; but path to config file can not be changed easily; also the name of the main SI system can not be changed easily; these issues will be discussed in detail in chapter 5 on the following page

Be robust and have a good error checking system

Is the system stable?	success	the system will report if data files or input are invalid but it is very difficult to let this happen unless data files are corrupt or do not comply with DTDs
Is there an error checking system?	success	at any point of execution, the system tries to see if any error have occurred and stop is this is the case

Chapter 5

Conclusions

Instead of recapping the main results of the project which are summarized in the abstract and in the introduction, we would like to concentrate on unsolved problems and future work.

Unsolved problems

Several minor problems was been detected in the course of extensive testing of the system. In particular, the user interface is generally easy to use, but problems might occur if many units/systems are to be added. The interface simply becomes full of new items.

The system is generally easy to configure but some configuration parameters can not be changed easily. Paths to CD and STS directories can be configured by editing "locations.cfg" file; but path to config file can not be changed easily. This would require to search and replace the path by running several scripts.

It is not easy to customize the name of the SI system. We will have to go through several scripts to replace the name of the SI CD. This is not a big problem because the name of the SI unit file does not change very often. In fact, as we will discuss below, such change has been proposed to the OpenMath community. If it is is accepted it is unlikely that the name will be changed again in the future. Both these points are of minor importance and do not affect at all the proper functioning of the developed system as well as the ease of its installation and use.

It is also worth mentioning that some composed units are impossible to represent using current user interface. For example we can not work with kilometres per hour, cubic metres.. One might argue that this is a user interface problem, but is it really the case?

OpenMath unit CD structure only allows to customize units by composing various units and prefixes. Yes, it is true that any custom unit can be represented using OpenMath but a new definition has to be created before being able to handle this custom unit. The designed system is a signature-based unit converter and can only convert between units found in signatures. Choosing a different time unit for composed measures like speed (for example to transform metre per second to kilometre per hour) is equivalent to introducing a new unit itself. For the same reason, various other popular units such as cubic metre can not be converted to and from using the current converter. One might think that this is the interface problem in considering the time as another form of prefix, but it is not actually the case and it can not be considered as an interface bug.

In addition, *arith1* and *interval1* subroutines emulate operations performed by the "arith1" and "interval1" CDs respectively. The problem might appear when someone decides to add a new operation to these CDs. This would not be reflected in the work of the developed unit converter. This requires *arith1* or *set1interval1* subroutines to be amended to include this new operation. However, this will not pose a big problem since "arith1" and "interval1" CDs are stored in the "Core" section of CDs on the OpenMath website. This means that it is very unlikely that they are going to improvements change in a near future. We can think of operations defined in the subroutines as a complete set which is enough to define any kind of unit representation.

Future work

While completing this project, we would like to discuss new improvements that can be made in the future. future First of all, the system can be improved by addressing the problems discussed above. In particular, we would like to spend more attention to interface design. Although the major emphasis of this project was in working within OpenMath concepts and user interface was of secondary importance, more advanced graphical user interface is highly desirable. The current interface is satisfactory but it will become difficult to use once many more units and systems are added to the converter.

The special attention is required to address the issue of configuration of the converter. As it was mentioned above, the path to the configuration file can be declared as a global Perl constant that is is available across all subroutines. This solutions reduces the configuration change to the change of the path to the configuration file to be made in the predefined place of the system. Name of the SI system CD can also be stored as a Perl constant and can be accessed by all subroutines.

Finally, the speed of the current system is quite satisfactory but could be improved more by processing two prefixes at once. This will allow the prefix CD to be opened and parsed once, thus saving memory and time. This effect can not be noticed on a single user system, but this can become more noticeable in the server environment when many users want to access the system simultaneously.

The error handling is implemented well in the system. But an even better error processing can be achieved by using CG::Carp Perl module.

Looking back at the beginning of the project I would spend more time by thorough analysis and development of more consistent and complete set of unit definitions. This would definitely increase the final product quality. In fact, this issue is identified as extremely important by OpenMath community and the appropriate work is currently under way. The content dictionaries for units are still at the early development stage and more effort and team work is required in this area.

A message to the OpenMath community

In course of this work, several changes and additions have been made to CD and STS related to processing units. They are reflected in the bug track report and listed in appendix C. While the changes made are quite obvious there are still several questions related to OpenMath unit data representation that appeared during the development process. I decided to adhere the existing OpenMath standards because my questions and concerns require an additional research and discussion by the OpenMath community. Nevertheless, it is useful to formulate these questions in the hope that they might be by OpenMath community at some point.

1. We agree with [4, text after] that "units_metric1" should be renamed into "units_si1". It would be very constructive to implement this change. If this were to happen then it would be highly desirable to make another CD , "units_metricmisc1", that will include the Metric *horsepower*, *hectare*, and other units as it is suggested in [4]. If renaming will take place, then "units_si1" should contain only *base SI* units. This point is not stressed in the article [4] but we suggest to move some units already defined in "units_metric1" to "units_metricmisc1", including *litre_pre1964*, *degree_Kelvin*, etc. Units that are not likely to be used by other CDs to define other units should go into "units_metricmisc1".

2. There are some more or less fundamental units that are not present in the current unit CDs. In particular, they include *inch*, *ounce*, *gallon*. As far as the volume units is concerned, there is no representation of solid volumes (e.g. cubic metre, cubic foot, etc). There are only liquid volumes defined. No clear distinction exists between liquid and solid volumes in Metric system, but lack of solid unit representations does make volume conversions limited.

3. Content of *CMP* in CDs is extremely misleading. It is OK for simple units but it poses the problems in case of units such as like *degree_Fahrenheit*. Here is a *CMP* content of the *degree_Fahrenheit*:
`<CMP> 1 degree Fahrenheit = 0.5556*(1-32) degrees Celsius </CMP>`
Obviously, it might be a good idea to reconsider *CMP* definitions for unit CDs. This is not a substantial change that might confuse any existing

software based on unit CDs. But since this Commented Mathematical Property objects, created by mathematicians, the most logical people by profession in the world, it is highly desirable to make the logic in this area even better.

Final words

II consider this project as highly successful and rewarding. To my opinion, the main objective has been achieved. As it was demonstrated by the testing report (see chapter 4 on page 44) the 99% of items from the system requirements have been fulfilled. It is my project supervisor who will decide the degree of my success in the development of the OpenMath unit converter. However, the brief conclusion regarding the purpose and meaning of this work can be made. The proposed extension to OpenMath that aims at attaching units to quantities does work extremely well and will greatly benefit the community by creating novel and useful system tools within OpenMath framework. Although the converter developed in the course of this project is just a small and very simple example of this type of applications, it demonstrates the power of this this extension that can enormously benefit the other areas of practical importance.

Demonstration of the system

A working demo of the designed OpenMath converter can be accessed at:`http://ccpc-jh2.bath.ac.uk/` (only accessible within University of Bath campus network). Source code is available upon request and is also enclosed in the form of CD to the hardcopy of this dissertation.

Bibliography

[1] U.S. Coast and 1893 Geodetic Survey Bulletin 26, April 5.

[2] James H. Davenport. On writing openmath content dictionaries. *ACM SIGSAM Bulletein*, 34(2):12–15, 2000.

[3] James H. Davenport. A small openmath type system. *ACM SIGSAM Bulletein*, 34(2):16–21, 2000.

[4] James H. Davenport. Units and dimensions in openmath. *ACM SIGSAM Bulletein*, 2003.

[5] M. Dewar. Openmath: An overview. *ACM SIGSAM Bulletein*, 34(2):2–5, 2000.

[6] http://crystal.win.tue.nl/products/openmath/lib/. Riaca openmath library website.

[7] http://en.wikipedia.org. Wikipedia, the free encyclopedia.

[8] http://expat.sourceforge.net. Expat website.

[9] http://java.sun.com/xml/jaxp/. Java api for xml processing (jaxp) website.

[10] http://pyxml.sourceforge.net. Pyxml, xml parsing module for python.

[11] http://sax.sourceforge.net/. Sax standard website.

[12] http://search.cpan.org/ coopercl/XML Parser/Parser.pm. Xml::parser perl module website.

[13] http://uk.php.net/domxml. Php5 dom extension website.

[14] http://www.convert me.com/en/convert/weight. Convertme.com: weight and mass conversion.

[15] http://www.openmath.org. Openmath standard website.

[16] http://www.w3.org/DOM/. Document object model (dom) standard website.

[17] http://www.w3.org/Math/. Mathml standard website.

[18] http://www.w3.org/XML/. Xml standard website.

[19] http://www.xmlsoft.org. Libxml2 parser website.

[20] http://xml.apache.org/. Xerces xml parsing modules website.

[21] Eliotte Rusty and W. Scott Means. *XML in a Nutshell.* O'Reilly, 2nd edition edition, 2002.

[22] S.Buswell, O.Caprotti, D.P.Carlisle, M.C.Dewar, M.Gaetano, and M.Kohlhase. *The OpenMath Standard*, 2.0 public 5a (7 april 2004) edition, 2004.

Appendix A

Development notes

Development notes include everything that could not be included in the main project write-up. These informal notes provide an in-depth view how the project was developed, what difficulties were faced (e.g. technical problems, etc). Although not completely relevant to the major goal of the project, these notes provide important information for someone two wants to understand how decisions were made and how technical problems were solved. This documents my own personal experience of software project development and management.

A.1 Preparation

Before I started to write specifications for the project, I carefully analyzed all the important stages of the project. However, several important issues required some input from my supervisor, Professor James Davenport. Therefore, I arranged a series of meetings with him to discuss the issues related to strategic goals of the project as well as some technical details. Here is a brief script of our discussions.

First meeting:

- Q: Is there a way to ensure a legality of a conversion, that is to ensure that converting from metres to miles is legal but converting from litres to centimetres is illegal?
  A: Small OpenMath Type System (STS files) will assist you in solving this problem.
- Q: Some non *base SI* units are not defined in terms of other units (for example, *bar*, standard imperial measure for pressure is not defined in terms of anything else but it is non SI unit!). How do I have to deal with

this situation?
A: The CDs on OpenMath official website are a little out of date and require updating. This can be considered as a bug and should be checked and corrected by OpenMath community.

- Q: There is no definition of kilometre at all. How do I have to deal with this?
  A: Use a *prefix* CD.

Second meeting:

- Q: Do I need to include project proposal and literature review in my project write-up?
  A: Proposal is left as a separate section of project write-up, literature review is integrated into the write-up.
- Q: Is it worth using RIACA library which is written in Java or can I use other alternative libraries that are much better interfaced with Perl?
  A: It doesn't matter what library you use, if you feel more confident and fluent with Perl then use some Perl library.

Third meeting:

- Q: How do we deal with conversions to and from "months" and similar units? There is no clear definitions of a month as it can be a time period consisting of 28, 29, 30 and 31 days (which is represented as a period in OpenMath).
  A: When converting this unit, return a period instead of a single value.
- Q: There are several parts of CDs and STSs that I don't consider as logical and consistent. They might be simply some bugs and there might appear a situation when these bugs could be eliminated by correcting appropriate CDs. Would you be able to give me latest versions of CDs or comment on my concerns and disagreements?
  A: The versions that are uploaded to OpenMath.org are the latest ones so the problems that you spotted do in fact exist. I will correct these as soon as possible and will send you the new versions. (Later, I added some bugs found in CDs to my bug tracking system.)

A.2 Software installation

An important issue I had to address is to to set up an adequate computational infrastructure for this project. This included the necessary prerequisites in terms of standard software, compilers, libraries, platform independence and compatibility. As discussed in section 3.2.4 on page 21, I have chosen to use Perl

with XML::DOM module. Perl is a part of standard UNIX installations, and for MS Windows there is a freely available installer (that can be downloaded from `http://www.activestate.com`). XML::DOM module does not come with Perl installations by default so it has to be installed separately. I checked with BUCS server system administrator about the possibility of installing XML::DOM module and make it available for other users. This would allow my project advisor as well as other interested users to have an access to my OpenMath unit converter. Due to security concerns BUCS system administrator was not able to install the third party XML::DOM module Perl module with system wide access permissions and advised to install it in my home directory.

A.2.1 Installing on UNIX (OS X)

The module of my choice (XML::DOM) is based on the Expat C++ library in requires several prerequisites They include:

- XML::RegExp
- XML::LibXML
- XML::Parser (at least v.2.28, 2.30 is required)
- LWP (libwww module is required)

XML::Parser requires installation of Expat C++ module that can be downloaded from `http://epxat.sourceforge.net`. Latest source has been downloaded and installed by extracting everything from archive, moving to the directory containing extracted files and running:

./configure
make
make install

All Perl modules were installed by downloading them from `http://www.cpan.org`, extracting files from module archives, locating module directory and running the following commands from there:

perl Makefile.pl
make
make test
make install (provided "make test" passed without errors).

When installing XML::Parser, Makefile.pl has to be run with two flags to specify the custom paths to "libexpat" and "expat.h" from the Expat C++ library.

Finally, LWP module requires installation of the following modules:

- libwww

- MIME::Base64
- URI
- HTML::Parser
- libnet
- Digest::MD5

I tested the required software infrastructure using my own Mac computer that runs Mac OS X (which is fully compatible UNIX operating system based on FreeBSD).

A.2.2 Installing on Windows

As mentioned in Resources section 3.2.4 on page 21, I decided to have my OpenMath system convertor running on MS Windows PCs for the purpose of public demonstration. The installation of everything required to get the system up and running was extremely easy. I downloaded and ran ActivePerl installer. After that I downloaded Perl Package Manager (ppm from command line) and typed the command: "install XML-DOM". This automatically installed the module of my choice for processing XML data, XML::DOM and all it's pre-requisites. After these easy to do steps, the system was up and running on MS Windows PC.

A.3 Content Dictionaries

It was not easy to get fluency with the structure of the CD archive. The major difficulty was to decide which versions of CDs to use and what are the main locations for these CDs. Although there is a separate CD page with the three subsections (Core, Public, Extra), there was no index page that would just list all files from the selected directory. Some CDs contained errors and turned out to be out of date. Later, I received updated versions of those from my project supervisor.

A.4 User interface

Before embedding HTML markup in the program code, I decided to produce a draft of the design as a sample HTML page. Then, I could do appropriate adjustments to get rid of some minor problems in layout. Once I was happy with the created design, I could use parts of HTML sample code in the script that produces the user interface.

A.5 Algorithms

A.5.1 Signature files

Signature files were the first first objects I started to work with. The first task was to parse XML in a sensible manner that is best suitable for my project. The first algorithm for extracting unit type information from STS files was built using many nested "if" loops to check for the right pattern of element tags. This approach not only introduced unnecessary code complications but also assumed that the structure of signature files is uniquely defined. Slight diversions from the structure would make the program fail to extract information.

As a result of mastering programming techniques,a second version of the algorithm was developed that was based on processing children recursively by a specially created subroutine. To ensure that element tags are occurred in correct order the subroutines have several parameters passed (father element names and attribute values). This simplified the code a lot, greatly reduced the indentation (nesting) volume and ensured that any valid STS file would be processed fully and correctly. This algorithm is described in more detail in the main project write-up in section 3.3.2 on page 25.

A.5.2 Parsing CDs

Prefix processing was done before the unit conversion part of the code. Therefore, I had to produce the code for opening and parsing of CDs and for jumping to the <FMP> tag of the needed object in a particular CD. Later it was discovered that the similar code was required in the unit processing section of the program so these two parts were extracted into separate subroutines in *UnitConverter.pm*.

A.5.3 Doing conversions

When thinking about the conversion algorithms, it was set out to produce a conversion type classification scheme that would then assist in creating the best possible algorithms for unit conversions. Initially, the classification was done by listing all possible types of conversions. This turned out to be a very bad approach that was really confusing and so another classification method has been used. The old classification scheme is listed below.

1. Same system conversions
   (a) base to non-base
   (b) non-base to base

   (c) non-base to non-base

2. Non-SI unit to SI unit conversions

   (a) base non-SI to non-base SI
   (b) non-base non-SI to base SI
   (c) non-base non-SI to non-base SI

3. SI unit to non-SI unit conversions

   (a) base SI to non-base non-SI
   (b) non-base SI to base non-SI
   (c) non-base SI to non-base non-SI

4. Non-SI unit to non-SI unit

   (a) base non-SI to non-base non-SI
   (b) non-base non-SI to base non-SI
   (c) non-base non-SI to non-base non-SI

Appendix B

User manual

One of the requirements of this OpenMath converter system was "ease of use". This requirement has successfully been met. As a result the interface is intuitive to use. Consequently, the user manual for this system is extremely short.

B.1 To install the system

Before installing, make sure that you have a web server, Perl and XML::DOM module installed.

To install the system, unpack the unit converter archive (can be found on the CD that is attached to this dissertation) into a directory that is configured to be accessible from the browser by the web server. Configure "dev" subdirectory to be a CGI directory on the web server. Set executable permissions for all files within "dev" directory apart from "locations.cfg". Installation is now complete and you can access the system using your browser. The address that you type into the address bar of a browser depends on how you configured your web server to process the unit converter directory. Note that if you have SSI enabled on the web server, then you can load the converter by accessing the main extraction directory from a web server. If SSI is not enabled, then you need to access the converter using the following address: `http://path_to_unit_converter_dir/dev/show.pl`.

Pre-installed version of this system can be accessed at: `http://ccpc-jh2.bath.ac.uk/`

B.2 To perform a unit conversion

Use your browser open a Unit Converter main page, located online at: `http://ccpc-jh2.bath.ac.uk/` (can only be accessed within the University campus network). The page should come up listing interfaces for all possible measure conversions. To browse to the specific measure section, just find the measure you are looking for in the menu at the top (labeled as "Convert:") and click on the appropriate measure type.

Once you are at the required measure section, type the amount of the original unit in the "Amount" input box. Then choose original and output prefixes and units using provided listboxes that are labeled appropriately. Once you have selected the correct values in listboxes, just click "Convert" button to perform the desired conversion. New page will appear showing results of the selected conversion.

B.3 To add a new unit

To add a new unit that will be available in the unit converter, two steps has to be performed:

1. Add the signature for the new unit to the appropriate STS file. Guidelines on this process are given in [22] and in [3];
2. Add the definition for this unit to the appropriate CD file. Guidelines on this process are given in [22] in [4] and in [2];

New units should appear automatically in the appropriate measure section if the above steps have been done and the additions comply with XML and OpenMath DTDs for CDs and STSs. Note that in order to see the changes, you have to refresh an interface page in browser. Also note that CDs and STSs that are modified must be in the configuration directory path. Otherwise the converter will not know about the existence of these CDs and STSs.

B.4 To add a new system

To add a new measure system to the unit converter, e.g. "units_us1" you have to perform the following steps:

1. Create new CD and STS files for the new measure system;

2. Put newly created files in the directory that is referenced by the converter for CDs and STSs (i.e. put new files in the directories specified by the configuration file, "locations.cfg");

If new CDs and STSs are valid, the system will find them and add all units defined by them into appropriate measure sections. Note that changes will not take effect until the interface page is refreshed in the browser.

B.5 To change the location of unit definition files

To change the paths to either the directory that stores all STS files or to the directory that stores all CD files, edit these paths in the configuration file, "locations.cfg".

B.6 To view the available CD and units information

Information on available CDs and units defined by them is displayed on the information page. This page can be accessed by clicking on the "Information" link which is at the top of the main interface page. Information page will also show the version, author and credits information.

Appendix C
Development testing - bug reports

Next few pages will list the bugs that have been found in both the system and data (CD and STS) files during the development process. These pages are extracts from the bug tracking system. The bug tracker can be accessed online at: http://www.bath.ac.uk/~ma1eo/mantis/ (login: guest, password: guest)

```
Data category
=============

======================================================================
http://www.bath.ac.uk/~ma1eo/mantis/view_bug_page.php?f_id=0000058
======================================================================
Reporter:                   eugene
Handler:                    eugene
======================================================================
Project:                    OpenMath Unit Converter
Bug ID:                     0000058
Category:                   Data
Reproducibility:            always
Severity:                   feature
Priority:                   none
Status:                     resolved
Resolution:                 fixed
======================================================================
Date Submitted:             05-11-04 14:34 BST
Last Modified:              05-11-04 14:34 BST
======================================================================
Summary:                    units_time1.ocd uses wrong CD
Description:
"units_time1" uses "relations1", which doesn't exist. It should instead
use "relation1".
======================================================================
 eugene - 05-11-04 14:34 BST
----------------------------------------------------------------------
Fixed now.

======================================================================
http://www.bath.ac.uk/~ma1eo/mantis/view_bug_page.php?f_id=0000043
======================================================================
Reporter:                   eugene
Handler:                    eugene
======================================================================
Project:                    OpenMath Unit Converter
Bug ID:                     0000043
Category:                   Data
Reproducibility:            always
Severity:                   feature
Priority:                   none
Status:                     resolved
Resolution:                 fixed
======================================================================
Date Submitted:             05-01-04 18:13 BST
Last Modified:              05-11-04 13:00 BST
```



```
======================================================================
Summary:                    Missing CD definitions
Description:
The newly proposed extension of OpenMath unit CDs states that unless  a
unit is  a "base  SI" unit,  it must  be defined  in terms of something
else.
If this is the case, then the following CDs still need to be defined:
+ litre_pre1964
+ metres_per_second
+ metres_per_second_sqrd
+ degree_Celcius or degree_Kelvin
+ pint
+ pound_mass
+ pound_force
+ degree_Fahrenheit
+ bar
======================================================================
 eugene - 05-01-04 18:45 BST
----------------------------------------------------------------------
Confirmed by JHD, but not decided what to do with litre_pre1964 and
degree_Celcius/degree_Kelvin

----------------------------------------------------------------------
 eugene - 05-09-04 19:58 BST
----------------------------------------------------------------------
Added an FMP for Fahreheit:

<CMP> 1 degree_Fahrenheit = 0.5556*(1-32) degrees_Celcius </CMP>
<FMP><OMOBJ>
  <OMA>
    <OMS name="eq" cd="relation1"/>
    <OMA>
      <OMS name="times" cd="arith1"/>
      <OMI> 1 </OMI>
      <OMS name="degree_Fahrenheit" cd="units_imperial1"/>
    </OMA>
    <OMA>
      <OMS name="times" cd="arith1"/>
      <OMF> 0.5556 </OMF>
      <OMA>
        <OMS name="minus" cd="arith1"/>
        <OMS name="degree_Celsius" cd="units_metric1"/>
        <OMI> 32 </OMI>
      </OMA>
    </OMA>
  </OMA>
</OMOBJ></FMP>
</CDDefinition>

----------------------------------------------------------------------
 eugene - 05-11-04 11:49 BST
----------------------------------------------------------------------
Added the following signature for the litre_pre1964:

<FMP><OMOBJ>
  <OMA>
    <OMS name="eq" cd="relation1"/>
```

```
    <OMA>
      <OMS name="times" cd="arith1"/>
      <OMI> 1 </OMI>
      <OMS name="litre_pre1964" cd="units_metric1"/>
    </OMA>
    <OMA>
      <OMS name="times" cd="arith1"/>
      <OMF> 1.000 028 </OMF>
      <OMS name="litre" cd="units_metric1"/>
    </OMA>
  </OMA>
</OMOBJ></FMP>
```

This complies with the definition given in the CMP.

---

eugene - 05-11-04 11:55 BST

The following FMP has been added for metric speed:

```
<FMP><OMOBJ>
<OMA>
  <OMS name="eq" cd="relation1"/>
  <OMS name="metres_per_second" cd="units_metric1"/>
  <OMA>
    <OMS name="divide" cd="arith1"/>
    <OMS name="metre" cd="units_metric1"/>
    <OMS name="second" cd="units_time1"/>
  </OMA>
</OMA>
</OMOBJ></FMP>
```

This is fully inline with imperial speed.

---

eugene - 05-11-04 11:59 BST

The following FMP has been added to "units_metric1" for acceleration:

```
<FMP><OMOBJ>
<OMA>
  <OMS name="eq" cd="relation1"/>
  <OMS name="metres_per_second_sqrd" cd="units_metric1"/>
  <OMA>
    <OMS name="divide" cd="arith1"/>
    <OMS name="metres_per_second" cd="units_metric1"/>
    <OMS name="second" cd="units_time1"/>
  </OMA>
</OMA>
</OMOBJ></FMP>
```

This is fully inline with acceleration defined in "units_imperial1".

---

eugene - 05-11-04 12:59 BST

As far as the decision about the SI base unit for temperature goes, I

think we should choose "degree_Celcius" for the following reason:
It is more sensible to express units from other systems in terms of base SI units. This is exactly what happened with "degree_Fahrenheit". Most (if not all) scientific texts define "degree_Fahrenheit" in terms of "degree_Celcius", so it is fair to think that the world has chosen the base SI unit to be "degree_Celcius".
It is important to note that scientists use "degree_Kelvin" most of the time as it is an absolute scale of temperature based on laws of heat rather than the freezing/boiling-points of water.
We therefore leave the deinition of "degrees_Celcius" untouched, but add the following FMP section to "degree_Kelvin":

```
<FMP><OMOBJ>
  <OMA>
    <OMS name="eq" cd="relation1"/>
    <OMA>
      <OMS name="times" cd="arith1"/>
      <OMI> 1 </OMI>
      <OMS name="degree_Kelvin" cd="units_metric1"/>
    </OMA>
    <OMA>
      <OMS name="minus" cd="arith1"/>
      <OMS name="degree_Celsius" cd="units_metric1"/>
      <OMF> 273.15 </OMF>
    </OMA>
  </OMA>
</OMOBJ></FMP>
```

---

eugene - 05-11-04 12:32 BST

The following FMP definition has been added to "pint" signature in "units_imperial1":

```
<FMP><OMOBJ>
  <OMA>
    <OMS name="eq" cd="relation1"/>
    <OMA>
      <OMS name="times" cd="arith1"/>
      <OMI> 1 </OMI>
      <OMS name="pint" cd="units_imperial1"/>
    </OMA>
    <OMA>
      <OMS name="times" cd="arith1"/>
      <OMF> 0.568 </OMF>
      <OMS name="litre" cd="units_metric1"/>
    </OMA>
  </OMA>
</OMOBJ></FMP>
```

---

eugene - 05-11-04 12:41 BST

The following FMP definition has been added for "pound_mass" in "units_imperial1":

```
<FMP><OMOBJ>
```

```
  <OMA>
    <OMS name="eq" cd="relation1"/>
    <OMA>
      <OMS name="times" cd="arith1"/>
      <OMI> 1 </OMI>
      <OMS name="pound_mass" cd="units_imperial1"/>
    </OMA>
    <OMA>
      <OMS name="times" cd="arith1"/>
      <OMF> 453.59 </OMF>
      <OMS name="gramme" cd="units_metric1"/>
    </OMA>
  </OMA>
</OMOBJ></FMP>

------------------------------------------------------------------------
 eugene - 05-11-04 12:50 BST
------------------------------------------------------------------------
The following FMP definition has been added for "pound_force" in
"units_imperial1":

<FMP><OMOBJ>
  <OMA>
    <OMS name="eq" cd="relation1"/>
    <OMA>
      <OMS name="times" cd="arith1"/>
      <OMI> 1 </OMI>
      <OMS name="pound_force" cd="units_imperial1"/>
    </OMA>
    <OMA>
      <OMS name="times" cd="arith1"/>
      <OMF> 4.448 </OMF>
      <OMS name="Newton" cd="units_metric1"/>
    </OMA>
  </OMA>
</OMOBJ></FMP>

------------------------------------------------------------------------
 eugene - 05-11-04 12:54 BST
------------------------------------------------------------------------
The following FMP definition has been added for "bar" in
"units_imperial1":

<FMP><OMOBJ>
  <OMA>
    <OMS name="eq" cd="relation1"/>
    <OMA>
      <OMS name="times" cd="arith1"/>
      <OMI> 1 </OMI>
      <OMS name="bar" cd="units_imperial1"/>
    </OMA>
    <OMA>
      <OMS name="times" cd="arith1"/>
      <OMI> 100000 </OMI>
      <OMS name="Pascal" cd="units_metric1"/>
    </OMA>
  </OMA>
```



```
</OMOBJ></FMP>

------------------------------------------------------------------------
 eugene - 05-11-04 13:00 BST
------------------------------------------------------------------------
All necessary non-base SI units are now correctly defined.

========================================================================
http://www.bath.ac.uk/~ma1eo/mantis/view_bug_page.php?f_id=0000057
========================================================================
Reporter:                   eugene
Handler:                    eugene
========================================================================
Project:                    OpenMath Unit Converter
Bug ID:                     0000057
Category:                   Data
Reproducibility:            always
Severity:                   feature
Priority:                   none
Status:                     resolved
Resolution:                 fixed
========================================================================
Date Submitted:             05-10-04 21:30 BST
Last Modified:              05-10-04 21:30 BST
========================================================================
Summary:                    Typo in units_imperial1.ocd
Description:
FMP for miles per hour squared includes the following line:
<OMS name="mile_per_hr" cd="units_imperial1"/>
mile_per_hr does not exist, the tag should be:
<OMS name="miles_per_hr" cd="units_imperial1"/>

========================================================================
 eugene - 05-10-04 21:30 BST
------------------------------------------------------------------------
Fixed now.

========================================================================
http://www.bath.ac.uk/~ma1eo/mantis/view_bug_page.php?f_id=0000056
========================================================================
Reporter:                   eugene
Handler:                    eugene
========================================================================
Project:                    OpenMath Unit Converter
Bug ID:                     0000056
Category:                   Data
Reproducibility:            always
Severity:                   feature
Priority:                   none
Status:                     resolved
Resolution:                 fixed
========================================================================
Date Submitted:             05-09-04 15:37 BST
Last Modified:              05-09-04 20:05 BST
========================================================================
Summary:               Litre definition in units_metric1 might be wrong
Description:
```

The current FMP definition is:

```
<FMP><OMOBJ>
<OMA>
  <OMS name="eq" cd="relations1"/>
  <OMA>
    <OMS name="times" cd="arith1"/>
    <OMI> 1000 </OMI>
    <OMS name="litre" cd="units_metric1"/>
  </OMA>
  <OMA>
    <OMS name="times" cd="arith1"/>
    <OMI> 1 </OMI>
    <OMA>
      <OMS name="power" cd="arith1"/>
      <OMS name="metre" cd="units_metric1"/>
      <OMI> 3 </OMI>
    </OMA>
  </OMA>
</OMA>
</OMOBJ></FMP>
```

This is a perfectly valid OpenMath, but it is not consistent with other definitions. All other similar FMP definitions express "1 unit is equal to something". Litre definition uses "1000 units is equal to something". Therefore I would suggest changing the Litre definition to comply with the common format, so it would now be:

```
<FMP><OMOBJ>
<OMA>
  <OMS name="eq" cd="relations1"/>
  <OMA>
    <OMS name="times" cd="arith1"/>
    <OMI> 1 </OMI>
    <OMS name="litre" cd="units_metric1"/>
  </OMA>
  <OMA>
    <OMS name="times" cd="arith1"/>
    <OMF> 0.001 </OMF>
    <OMA>
      <OMS name="power" cd="arith1"/>
      <OMS name="metre" cd="units_metric1"/>
      <OMI> 3 </OMI>
    </OMA>
  </OMA>
</OMA>
</OMOBJ></FMP>
```

```
======================================================================
 eugene - 05-09-04 20:05 BST
----------------------------------------------------------------------
Changed now.

======================================================================
http://www.bath.ac.uk/~ma1eo/mantis/view_bug_page.php?f_id=0000055
======================================================================
Reporter:                   eugene
Handler:                    eugene
======================================================================
```



```
Project:                    OpenMath Unit Converter
Bug ID:                     0000055
Category:                   Data
Reproducibility:            always
Severity:                   feature
Priority:                   none
Status:                     resolved
Resolution:                 fixed
======================================================================
Date Submitted:             05-09-04 15:07 BST
Last Modified:              05-09-04 20:04 BST
======================================================================
Summary:                    Typo in units_metric1.ocd
Description:
1) In definition of "litre_pre1964", typo:
"A lire is, since 1901", should say "A litre is, since 1901".
Not critical, does not affect correct functioning.
======================================================================
 eugene - 05-09-04 20:04 BST
----------------------------------------------------------------------
Fixed.

======================================================================
http://www.bath.ac.uk/~ma1eo/mantis/view_bug_page.php?f_id=0000054
======================================================================
Reporter:                   eugene
Handler:                    eugene
======================================================================
Project:                    OpenMath Unit Converter
Bug ID:                     0000054
Category:                   Data
Reproducibility:            always
Severity:                   feature
Priority:                   none
Status:                     resolved
Resolution:                 fixed
======================================================================
Date Submitted:             05-05-04 13:14 BST
Last Modified:              05-09-04 20:03 BST
======================================================================
Summary:                Missed value in html version of units_imperial1
Description:
Definition of "foot" reads:
eq (times ( 1 , foot) , times ( , metre) )
this should be:
eq (times ( 1 , foot) , times ( 0.3048 , metre) )
This is not affecting the original ocd version of the file which
doesn't have this error, so no need to worry about this for the
project. Only inform JHD about this.
======================================================================
 eugene - 05-05-04 13:15 BST
----------------------------------------------------------------------
Informed JHD and confirmed by him.
----------------------------------------------------------------------
 eugene - 05-09-04 20:03 BST
----------------------------------------------------------------------
Will be fixed in next html version.
```

```
======================================================================
http://www.bath.ac.uk/~ma1eo/mantis/view_bug_page.php?f_id=0000046
======================================================================
Reporter:                   eugene
Handler:                    eugene
======================================================================
Project:                    OpenMath Unit Converter
Bug ID:                     0000046
Category:                   Data
Reproducibility:            always
Severity:                   feature
Priority:                   none
Status:                     resolved
Resolution:                 fixed
======================================================================
Date Submitted:             05-01-04 18:24 BST
Last Modified:              05-09-04 20:02 BST
======================================================================
Summary:                    units_ops.ocd uses wrong CD
Description:
units_ops.ocd uses relations1.ocd which doesn't exist. Should be using
relation1.ocd.
======================================================================
 eugene - 05-01-04 18:28 BST
----------------------------------------------------------------------
Confirmed by JHD.
----------------------------------------------------------------------
 eugene - 05-09-04 20:02 BST
----------------------------------------------------------------------
Resolved now.

======================================================================
http://www.bath.ac.uk/~ma1eo/mantis/view_bug_page.php?f_id=0000045
======================================================================
Reporter:                   eugene
Handler:                    eugene
======================================================================
Project:                    OpenMath Unit Converter
Bug ID:                     0000045
Category:                   Data
Reproducibility:            always
Severity:                   feature
Priority:                   none
Status:                     resolved
Resolution:                 fixed
======================================================================
Date Submitted:             05-01-04 18:22 BST
Last Modified:              05-09-04 20:01 BST
======================================================================
Summary:                    Typo in units_metric1.sts
Description:
"metre_sqd" should be  "metre_sqrd" and "metres_per_second_sqd"  should
be "metres_per_second_sqrd".
======================================================================
 eugene - 05-01-04 18:43 BST
----------------------------------------------------------------------
```



```
Confirmed by JHD.
----------------------------------------------------------------------
 eugene - 05-09-04 20:01 BST
----------------------------------------------------------------------
Fixed now.

======================================================================
http://www.bath.ac.uk/~ma1eo/mantis/view_bug_page.php?f_id=0000044
======================================================================
Reporter:                   eugene
Handler:                    eugene
======================================================================
Project:                    OpenMath Unit Converter
Bug ID:                     0000044
Category:                   Data
Reproducibility:            always
Severity:                   feature
Priority:                   none
Status:                     resolved
Resolution:                 fixed
======================================================================
Date Submitted:             05-01-04 18:17 BST
Last Modified:              05-09-04 20:00 BST
======================================================================
Summary:                    Typo in units_imperial1.ocd
Description:
FMP sections for miles_per_hr and miles_per_hr_sqrd have a typo in  OMS
tag  that  assigns  a  name  to  the  relation.  In  first  case  it is
"miles_per_h" but should be  "miles_per_hr_sqrd". In second case  it is
"miles_per_h" but should be "miles_per_hr_sqrd".
======================================================================
 eugene - 05-01-04 18:44 BST
----------------------------------------------------------------------
Confirmed by JHD.
----------------------------------------------------------------------
 eugene - 05-09-04 20:00 BST
----------------------------------------------------------------------
Fixed now.

======================================================================
http://www.bath.ac.uk/~ma1eo/mantis/view_bug_page.php?f_id=0000047
======================================================================
Reporter:                   eugene
Handler:                    eugene
======================================================================
Project:                    OpenMath Unit Converter
Bug ID:                     0000047
Category:                   Data
Reproducibility:            always
Severity:                   feature
Priority:                   none
Status:                     resolved
Resolution:                 fixed
======================================================================
Date Submitted:             05-01-04 18:27 BST
Last Modified:              05-09-04 15:48 BST
======================================================================
```

Summary: Velocity and speed are confused
Description:
units_metric1.sts defines "speed" but units_imperial1.sts defines "velocity". One or the other needs to be used for consistency.

---

eugene - 05-09-04 15:48 BST

Definitions:
Speed is the rate or a measure of the rate of motion, especially: Distance traveled divided by the time of travel.
Velocity is a vector quantity whose magnitude is a body's speed and whose direction is the body's direction of motion. [APP & SES 2003 & 2002]
Couldn't get feedback from JHD on this matter. Therefore, it is decided to go with Speed, because OpenMath objects only represent quantities, no direction.

System category

---

http://www.bath.ac.uk/~ma1eo/mantis/view_bug_page.php?f_id=0000060

Reporter: eugene
Handler: eugene

Project: OpenMath Unit Converter
Bug ID: 0000060
Category: System
Reproducibility: always
Severity: feature
Priority: none
Status: resolved
Resolution: fixed

Date Submitted: 05-11-04 17:16 BST
Last Modified: 05-11-04 17:17 BST

Summary: FindObject sub: wrong number of FMP tags
Description:
FindObject sub did not allow any object to have more than one FMP definition. This is wrong, e.g. year has two FMPs - one in terms of months and one in terms of days.

---

eugene - 05-11-04 17:17 BST

Fixed. Converter will now use the first FMP definition.

---

http://www.bath.ac.uk/~ma1eo/mantis/view_bug_page.php?f_id=0000059

Reporter: eugene
Handler: eugene

Project: OpenMath Unit Converter
Bug ID: 0000059

Category: System
Reproducibility: always
Severity: feature
Priority: none
Status: resolved
Resolution: fixed

Date Submitted: 05-11-04 15:50 BST
Last Modified: 05-11-04 17:14 BST

Summary: Input parser does not accept negative values
Description:
Input parser does not accept negative values, but there exist negative temperatures, so this needs to be allowed.

---

eugene - 05-11-04 17:06 BST

Fixed, but does not accept zero values now.

---

eugene - 05-11-04 17:14 BST

Fixed "zero" problem.

1) Modified the input processing to be:

```
my $amount = ($session->param('amount') == 0)?$session->param('amount'):'';
```

2) Added a check to "arith1" sub to: see if number is equal to "zero" then substitute real zero instead of that value

3) Added a check for zero denominator in divide operation

---

http://www.bath.ac.uk/~ma1eo/mantis/view_bug_page.php?f_id=0000048

Reporter: eugene
Handler: eugene

Project: OpenMath Unit Converter
Bug ID: 0000048
Category: System
Reproducibility: always
Severity: feature
Priority: none
Status: resolved
Resolution: fixed

Date Submitted: 05-01-04 18:58 BST
Last Modified: 05-03-04 15:41 BST

Summary: STS parsing: only prefix CDs parsed correctly
Description:
STS parsing: only prefix CDs parsed correctly.

This is likely to be because of the incorrect processing of OMS tags. Also, it seems that the first main hash(sysData) values are written at

the wrong time. These values should also be written only when processing OMS tags.
==
 eugene - 05-03-04 15:41 BST

Corrected now. All data is only written when doing the final 'if' statement (when processing OMS tags). This means that another parameter is passed to a subroutine ($passAttrib2) existance of which we check in this last 'if' statement. Also, we now only check $attrData{'cd'} has some value assigned to it rather than checking whether it equals to prefix cd.
Another 'if' stamenet is added inside this one to check that $passAttrib2 doesn't contain 'unit_' at the beginning.

==
http://www.bath.ac.uk/~ma1eo/mantis/view_bug_page.php?f_id=0000051
==
Reporter: eugene
Handler: eugene
==
Project: OpenMath Unit Converter
Bug ID: 0000051
Category: System
Reproducibility: always
Severity: feature
Priority: none
Status: resolved
Resolution: fixed
==
Date Submitted: 05-03-04 17:15 BST
Last Modified: 05-04-04 21:03 BST
==
Summary: Units not sorted in listboxes
Description:
Some values in unit listboxes are not sorted. This is because some units have first letter already in uppercase and we do sorting before uppercasing the first letter.
==
 eugene - 05-04-04 21:03 BST

Fixed now. Instead of pushing values into a scalar, push them into a hash, and then print sorted keys in hash.

==
http://www.bath.ac.uk/~ma1eo/mantis/view_bug_page.php?f_id=0000050
==
Reporter: eugene
Handler: eugene
==
Project: OpenMath Unit Converter
Bug ID: 0000050
Category: System
Reproducibility: always
Severity: feature
Priority: none
Status: resolved
Resolution: fixed

==
Date Submitted: 05-03-04 15:43 BST
Last Modified: 05-04-04 21:05 BST
==
Summary: Another check is needed in Startup.pm (Traverse sub)
Description:
To make the system even more robust, we need to add further check for a cd value in OMS and only accept those signatures that have a cd value of 'units_sts' or 'dimensions1' in OMS tag.
==
 eugene - 05-04-04 21:05 BST

Added now.

==
http://www.bath.ac.uk/~ma1eo/mantis/view_bug_page.php?f_id=0000049
==
Reporter: eugene
Handler: eugene
==
Project: OpenMath Unit Converter
Bug ID: 0000049
Category: System
Reproducibility: always
Severity: feature
Priority: none
Status: resolved
Resolution: fixed
==
Date Submitted: 05-03-04 15:08 BST
Last Modified: 05-04-04 21:05 BST
==
Summary: Problems with GetPrefixes sub in ExtractData.pm
Description:
Prefixes are checked using the wrong hash in ExtractData.pm (GetPrefixes sub). Instead of receiving a reference to sysDataRef hash, we should receive a reference unitDataRef and compare a value with 'unit_prefix' instead of 'units_siprefix1'.

The algorithm which is used now still works, but it will be more correct to do it using the way that is suggested above.
==
 eugene - 05-04-04 21:05 BST

Program uses correct hash and checks against correct value now.

==
http://www.bath.ac.uk/~ma1eo/mantis/view_bug_page.php?f_id=0000053
==
Reporter: eugene
Handler: eugene
==
Project: OpenMath Unit Converter
Bug ID: 0000053
Category: System
Reproducibility: always
Severity: feature

Priority: none
Status: resolved
Resolution: fixed

Date Submitted: 05-05-04 12:02 BST
Last Modified: 05-05-04 13:37 BST

Summary: Input parsing: amount input problems
Description:
The input parsing script accepts whol number values like 2 or 567, but doesn't accept values like 0.5 or 23.1.

eugene - 05-05-04 13:37 BST

Fixed now.
Was: if ($amount !~ /D/)
Now: if ($amount !~ /^-?d+.?d*$/ || $amount <= 0)

http://www.bath.ac.uk/~ma1eo/mantis/view_bug_page.php?f_id=0000052

Reporter: eugene
Handler: eugene

Project: OpenMath Unit Converter
Bug ID: 0000052
Category: System
Reproducibility: always
Severity: feature
Priority: none
Status: resolved
Resolution: fixed

Date Submitted: 05-04-04 17:05 BST
Last Modified: 05-09-04 13:34 BST

Summary: The input parser does not accept a value of 'none'
Description:
The input parser does not accept a value of none for prefix because this value is not present in the prefix list. But this value is valid since it means that no prefix should be used.

eugene - 05-09-04 13:34 BST

Fixed now.0

Appendix D□
Unit converter source code

Next few pages list the source code for the main converter functions used by □ the system.

```
package UnitConverter;
use XML::DOM;

BEGIN {

  # declare a hash that stores a CD name and a reference to the
  # XML document object for that CD name, every sub in this
  # module can access this hash
  our %cdDocuments;

}


################################################################################
# GetPrefixValue receives the prefix name and a reference to the sysData
# hash which stores CD names for each object (e.g. for prefixes as well).
# it then calculates and returns a value for that prefix
sub GetPrefixValue {
  my $prefixName = shift @_;
  my $sysDataRef = shift @_;

  if ( !$prefixName || ref($sysDataRef) ne 'HASH' ){
    print "<br><b>Error:</b> parameters passed to GetPrefixValue are
invalid";
    return;
  }

  # parse prefix CD and get a reference to document object back
  my $doc = ParseCd( $$sysDataRef{$prefixName} );

  # find the reference to the needed object
  my @prefixData = FindObject($doc, $prefixName);
  my $node = shift @prefixData;
  if ( !$node ){
    return;
  }
  # check that prefix definition exists
  if ( ref($node) ne 'XML::DOM::Element' ){
    print "<br><b>Error:</b> corrupt definition, no FMP found for prefix";
    return;
  }

  # ensuring that the begining of the prefix definition follows the rules
  if ( $node -> getNodeName() ne 'OMA' ){
    print "<br><b>Error:</b> corrupt FMP definition";
    return;
  }

  my @childNodes = $node -> getChildNodes();
```



```
  my @foundNodes;
  foreach $childNode (@childNodes){
    if ( $childNode -> getNodeType() == ELEMENT_NODE ){
      push(@foundNodes, $childNode);
    }
  }
  # ensuring that the prefix definition follows the rules
  if ( $foundNodes[0] -> getNodeName() ne 'OMS' || $foundNodes[1] ->
getNodeName() ne 'OMA' || $foundNodes[2] -> getNodeName() ne 'OMV' ){
    print "<br><b>Error:</b> corrupt FMP definition, looking for OMS, OMA &
OMV tags";
    return;
  }

  if ( ( $foundNodes[0] -> getAttributeNode("name") -> getValue() ) ne
'times' || ( $foundNodes[0] -> getAttributeNode("cd") -> getValue() ) ne
'arith1' ){
    print "<br><b>Error:</b> corrupt FMP definition, looking for 'times'
operation from 'arith1' CD";
    return;
  }
  if ( ( $foundNodes[2] -> getAttributeNode("name") -> getValue() ) ne
'unit' ){
    print "<br><b>Error:</b> corrupt FMP definition, looking for 'unit'
name";
    return;
  }

  @childNodes = $foundNodes[1] -> getChildNodes();
  $#foundNodes = -1;
  foreach $childNode (@childNodes){
    if ( $childNode -> getNodeType() == ELEMENT_NODE ){
      push(@foundNodes, $childNode);
    }
  }
  # ensuring that the prefix definition follows the rules
  if ( $foundNodes[0] -> getNodeName() ne 'OMS' || $foundNodes[1] ->
getNodeName() ne 'OMI' || $foundNodes[2] -> getNodeName() ne 'OMI' ){
    print "<br><b>Error:</b> prefix defined incorrectly";
    return;
  }
  # checking that prefix def uses arith1 to compose a value
  if ( ( $foundNodes[0] -> getAttributeNode("cd") -> getValue() ) eq
'arith1' || ( $foundNodes[0] -> getAttributeNode("name") -> getValue() ) ){
    my $operation = $foundNodes[0] -> getAttributeNode("name") ->
getValue();
    my $firstArg = $foundNodes[1] -> getFirstChild() -> getNodeValue();
    $firstArg =~ s/ //g;
    my $secondArg = $foundNodes[2] -> getFirstChild() -> getNodeValue();
    $secondArg =~ s/ //g;
    $value = arith1($operation, $firstArg, $secondArg);
    if ($value){
      return $value;
    }
    else {
      return;
    }
```

```
  }
  else {
    print "<br><b>Error:</b> prefix defined incorrectly, looking for
'arith1' reference";
    return;
  }

  return;
} # end GetPrefixValue


#############################################################################
# ConvertUnits does the main unit conversion algorithm
sub ConvertUnits {
  my $amount = shift @_;
  my $inUnitName = shift @_;
  my $outUnitName = shift @_;
  my $sysDataRef = shift @_;
  my $unitDataRef = shift @_;

  if ( !$amount || !$inUnitName || !$outUnitName || ref($sysDataRef) ne
'HASH' || ref($unitDataRef) ne 'HASH' ){
    print "<br><b>Error:</b> parameters passed to ConvertUnits are
invalid";
    return;
  }

  # getting parsed versions of CDs that contain input and output unit defs.
  my $inUnitDoc;
  # trying to avoid parsing if the CD has already been parsed
  if ( $cdDocuments{ $$sysDataRef{$inUnitName} } ){
    # get a reference to the parsed CD from the cdDocuments hash
    $inUnitDoc = $cdDocuments{ $$sysDataRef{$inUnitName} };
  }
  else {
    # parse CD and get a reference to document object back
    $inUnitDoc = ParseCd( $$sysDataRef{$inUnitName} );
  }
  my $outUnitDoc;
  # trying to avoid parsing if the CD has already been parsed
  if ( $cdDocuments{ $$sysDataRef{$outUnitName} } ){
    # get a reference to the parsed CD from the cdDocuments hash
    $outUnitDoc = $cdDocuments{ $$sysDataRef{$outUnitName} };
  }
  else {
    # parse CD and get a reference to document object back
    $outUnitDoc = ParseCd( $$sysDataRef{$outUnitName} );
  }

  # find the reference to the input object
  my @inUnitData = FindObject($inUnitDoc, $inUnitName);
  my $inUnitNode = shift @inUnitData;
  if ( !$inUnitNode ){
    return;
  }
  if ( ref($inUnitNode) ne 'XML::DOM::Element' && ( $inUnitName ne 'second'
&& $$sysDataRef{$inUnitName} ne 'units_metric1' ) ){
```



```
    print "<br><b>Error:</b> corrupt definition, no FMP found for the
$inUnitName unit which is not in units_metric1";
    return;
  }
  # check that the second child of OMA is an OMA or OMS tag
  if ( $inUnitNode != 1 && $inUnitNode -> getNodeName() ne 'OMA' &&
$inUnitNode -> getNodeName() ne 'OMS' ){
    print "<br><b>Error:</b> corrupt definition, GetObject did not detect
it for some reason";
    return;
  }
  # find the reference to the output object
  my @outUnitData = FindObject($outUnitDoc, $outUnitName);
  my $outUnitNode = shift @outUnitData;
  if ( !$outUnitNode ){
    return;
  }
  if ( ref($outUnitNode) ne 'XML::DOM::Element' && ( $outUnitName ne
'second' && $$sysDataRef{$outUnitName} ne 'units_metric1' ) ){
    print "<br><b>Error:</b> corrupt definition, no FMP found for the
$inUnitName unit which is not in units_metric1";
    return;
  }
  # check that the second child of OMA is an OMA or OMS tag
  if ( $outUnitNode != 1 && $outUnitNode -> getNodeName() ne 'OMA' &&
$outUnitNode -> getNodeName() ne 'OMS' ){
    print "<br><b>Error:</b> corrupt definition, GetObject did not detect
it for some reason";
    return;
  }
  ####################

  # classification of conversions here
  if ( $#outUnitData < 0 ){
    # see if input is SI. if yes, produce an error, if not - do a forward
conversion using input
    if ( $#inUnitData < 0 ){
      print "<br><b>Error:</b> one of the definitions corrupt, ConvertUnits
sub found two base SI units";
      return;
    }
    # do a forward conversion
    else {
      # class 1
      if ( $inUnitData[0] eq $outUnitName ){
        return ForwProcessOma($inUnitNode, $outUnitName, $amount);
      }
      # class 2
      else {
        my $newAmount = ForwProcessOma($inUnitNode, $inUnitData[0],
$amount);
        return ConvertUnits($newAmount, $inUnitData[0], $outUnitName,
$sysDataRef, $unitDataRef);
      }
    }
  }
  elsif ( $#outUnitData == 0 ){
```

```
    # only from base si to some unit uses backwards conversion
    # class 3 & 4
    if ( $#inUnitData < 0 ){
      return BackwProcessOma($outUnitNode, $inUnitName, $amount,
$sysDataRef, $unitDataRef);
    }
    # non-base to non-base
    else {
      # check to see if output is defined in terms of input
      # class 5 part 2
      if ( $outUnitData[0] eq $inUnitName ){
        #do a backward conversion
        return BackwProcessOma($outUnitNode, $inUnitName, $amount,
$sysDataRef, $unitDataRef);
      }
      else {
        # this case happens either if input is defined in terms of
        # output or of the two are not linked directly
        # class 5 part 1
        if ( $inUnitData[0] eq $outUnitName ){
          return ForwProcessOma($inUnitNode, $outUnitName, $amount);
        }
        # class 6
        else {
          my $newAmount = ForwProcessOma($inUnitNode, $inUnitData[0],
$amount);
          return ConvertUnits($newAmount, $inUnitData[0], $outUnitName,
$sysDataRef, $unitDataRef);
        }
      }
    }
  }
  # class 7
  else {
    # check if the input is also composite.
    if ( $#inUnitData > 0 ){
      # check that output and input are expressed using the same number of
terms
      if ( $#outUnitData == $#inUnitData ){
        # check that type of each pair is the same
        for ( $i=0; $i < ($#inUnitData + 1); $i++ ) {
          if ( $$unitDataRef{ $inUnitData[$i] } ne $$unitDataRef{
$outUnitData[$i] } ){
            print "<br><b>Error:</b> illegal set of definitions, each unit
pair for composite conversions must have the same type";
            return;
          }
        }
        return CompositeProcessOma($inUnitNode, $outUnitNode, $amount);
      }
      else {
        print "<br><b>Error:</b> illegal conversion, both input and output
need to have the same number of terms that define them";
        return;
      }

    }
```



```
    else {
      print "<br><b>Error:</b> illegal conversion, both input and output
need to be composite if at least one of them is";
      return;
    }
  }



} # end ConverUnits

##############################################################################
# This sub will convert two units provided one is expressed in terms of
# the other one
sub ForwProcessOma {
  # recieve a ref to tag
  # also receive name of the output unit and a value

  my $tagRef = shift @_;
  my $outUnitName = shift @_;
  my $amount = shift @_;

  if ( !$outUnitName || !$amount || ref($tagRef) ne 'XML::DOM::Element' ){
    print "<br><b>Error:</b> parameters passed to ForwProcessOma are
invalid";
    return;
  }

  if ( $tagRef -> getNodeName() eq 'OMA' ){

    # get all child nodes of OMA tag
    my @childNodes = $tagRef -> getChildNodes();
    # filter out only element nodes from the list of childnoes
    my @foundNodes;
    foreach $childNode (@childNodes){
      if ( $childNode -> getNodeType() == ELEMENT_NODE ){
        push(@foundNodes, $childNode);
      }
    }
    if ( $foundNodes[0] -> getNodeName() ne 'OMS' || $#foundNodes != 2 ){
      print "<br><b>Error:</b> first child of OMA tag should always be OMS
tag";
      return;
    }

    # call this sub again for second and third child
    my $firstValue = ForwProcessOma($foundNodes[1], $outUnitName, $amount);
    my $secondValue = ForwProcessOma($foundNodes[2], $outUnitName,
$amount);
    if ( !$firstValue || !$secondValue ){
      # no need to report an error as this must have been done already
      return;
    }

    # perform an op from first child on them
    #e.g. <OMS name="times" cd="arith1"/>
    if ( !( $foundNodes[0] -> getAttributeNode("name") -> getValue() ) || (
```

```
$foundNodes[0] -> getAttributeNode("cd") -> getValue() ) ne 'arith1' ){
      print "<br><b>Error:</b> corrupt FMP definition, looking for an
operation from 'arith1' CD";
      return;
    }

    my $opName = $foundNodes[0] -> getAttributeNode("name") -> getValue();

    # perform an op specified by the OMS tag
    my $result = arith1($opName, $firstValue, $secondValue);
    return $result;

  }
  elsif ( $tagRef -> getNodeName() eq 'OMS' ){
    if ( !( $tagRef -> getAttributeNode("name") ) ){
      print "<br><b>Error:</b> corrupt FMP definition, OMS tag doesn't have
a 'name' attribute";
      return;
    }

    # one unit is expressed in terms of the other unit
    if ( $tagRef -> getAttributeNode("name") -> getValue() eq $outUnitName
){
      return $amount;
    }

  }
  elsif ( $tagRef -> getNodeName() eq 'OMI' || $tagRef -> getNodeName() eq
'OMF' ){
    my $value = $tagRef -> getFirstChild() ->  getNodeValue();
    $value =~ s/ //g;
    return $value;
  }

  return;

} # end ForwProcessOma

#############################################################################
# This sub will convert two units provided one is expressed in terms of
# the other one, reversed conversion is done here
sub BackwProcessOma {
  # recieve a ref to tag
  # also receive name of the input unit and a value

  my $tagRef = shift @_;
  my $inUnitName = shift @_;
  my $amount = shift @_;
  my $sysDataRef = shift @_;
  my $unitDataRef = shift @_;

  if ( !$inUnitName || !$amount || ref($tagRef) ne 'XML::DOM::Element' ||
ref($sysDataRef) ne 'HASH' || ref($unitDataRef) ne 'HASH' ){
    print "<br><b>Error:</b> parameters passed to BackwProcessOma are
invalid";
    return;
  }
```



```
  if ( $tagRef -> getNodeName() eq 'OMA' ){
    # get all child nodes of OMA tag
    my @childNodes = $tagRef -> getChildNodes();
    # filter out only element nodes from the list of childnoes
    my @foundNodes;
    foreach $childNode (@childNodes){
      if ( $childNode -> getNodeType() == ELEMENT_NODE ){
        push(@foundNodes, $childNode);
      }
    }
    if ( $foundNodes[0] -> getNodeName() ne 'OMS' || $#foundNodes != 2 ){
      print "<br><b>Error:</b> first child of OMA tag should always be OMS
tag";
      return;
    }
    if ( !( $foundNodes[0] -> getAttributeNode("name") -> getValue() ) || (
$foundNodes[0] -> getAttributeNode("cd") -> getValue() ) ne 'arith1' ){
      print "<br><b>Error:</b> corrupt FMP definition, looking for an
operation from 'arith1' CD";
      return;
    }

    # now try to get values for two children, make sure that unit name is
not encountered
    my $firstValue = ForwProcessOma($foundNodes[1], 'none4535435',
$amount);
    my $secondValue = ForwProcessOma($foundNodes[2], 'none45353', $amount);
    if ( !$firstValue && !$secondValue ){
      print "<br><b>Error:</b> at least one child of OMA tag was expected
to have a value returned";
      return;
    }
    if ( $firstValue && $secondValue ){
      print "<br><b>Error:</b> both children of OMA tag return values which
is not acceptable";
      return;
    }

    # perform an op.
    if ( $firstValue ){

      # define op
      my $opName = $foundNodes[0] -> getAttributeNode("name") ->
getValue();
      if ( $opName eq 'plus' ){
        $amount = -$amount;
      }
      elsif ( $opName eq 'minus' ){
        $opName = 'plus';
        $amount = -$amount;
      }
      elsif ( $opName eq 'times' ){
        $opName = 'divide';
      }
      # divide op does not change things
```

```
      # perform an op specified by the OMS tag
      my $result = arith1($opName, $amount, $firstValue);

      # continue recursion
      return BackwProcessOma($foundNodes[2], $inUnitName, $result,
$sysDataRef, $unitDataRef);

    }
    elsif ( $secondValue ){

      # define op
      my $opName = $foundNodes[0] -> getAttributeNode("name") ->
getValue();
      if ( $opName eq 'plus' ){
        $amount = -$amount;
      }
      elsif ( $opName eq 'minus' ){
        $opName = 'plus';
      }
      elsif ( $opName eq 'times' ){
        $opName = 'divide';
      }
      elsif ( $opName eq 'divide' ){
        $opName = 'times';
      }

      # perform an op specified by the OMS tag
      my $result = arith1($opName, $amount, $secondValue);

      # continue recursion
      return BackwProcessOma($foundNodes[1], $inUnitName, $result,
$sysDataRef, $unitDataRef);

    }


  }
  # case when OMS tag
  elsif ( $tagRef -> getNodeName() eq 'OMS' ){
    if ( $tagRef -> getAttributeNode("name") -> getValue() eq $inUnitName
){
      # case when should return the value together with applying op to it
      return $amount;
    }
    else {
      # case when this is not the needed unit
      return ConvertUnits($amount, $inUnitName, $tagRef ->
getAttributeNode("name") -> getValue(), $sysDataRef, $unitDataRef);
    }
  }
  else {
    print "<br><b>Error:</b> corrupt FMP definition, OMA or OMS tag
expected, BackwProcessOma sub";
    return;
  }
```



```
} # end BackwProcessOma

##############################################################################
# ParseCd receives a CD name, tries to open this CD, parse it with XML
# parser and return a reference to the document object for that CD
sub ParseCd {
  my $cdName = shift @_;

  # making sure that cdName is passed to this sub
  if ( !$cdName ){
    print "<br><b>Error:</b> no parameters passed to ParseCd function";
    return;
  }

  # path to config file
  my $cfigFile = "locations.cfg";
  # checking that the config file exists and opening it
  if ( !open(CONFIG, $cfigFile) ){
    print "<br><b>Error:</b> can't open config file";
    return;
  }
  # putting config file data into the array
  my %paths;
  my ($pathName, $pathValue);
  while ( <CONFIG> ) {
    chomp;
    ($pathName,$pathValue) = split(/::/,$_);
    if($pathName && $pathValue) { $paths{$pathName} = $pathValue; }
  }
  close(CONFIG);
  if( !$paths{'cd'} || !$paths{'sts'} ){
    print "<br><b>Error:</b> no paths for either CD or STS directories,
check that config file has the correct paths";
    return;
  }

  my ($doc, $node);

  # parse the prefix file
  my $parser = new XML::DOM::Parser;
  eval { $doc =  $parser -> parsefile($paths{'cd'} . "/" . $cdName .
".ocd") };
  # catching "exceptions" here, in case invalid xml file
  if ($@) {
    print "<br><b>Error:</b> $@";
    return;
  }

  # put an entry into our global hash so that same CDs are only parsed
  # once
  $cdDocuments{$cdName} = $doc;
  return $doc;

} # end ParseCd

##############################################################################
# FindObject finds a reference to the first <OMA> tag of the object
```

```
# definition given a reference to the document object of a CD and a name
# of the object, OMA tag of which is required
sub FindObject {
  my $doc = shift @_;
  my $objectName = shift @_;
  if ( !$doc || !$objectName ){
    print "<br><b>Error:</b> not enough parameters passed to FindObject
function";
    return;
  }

  # find the section of CD which is responsible for representing our object
  my @foundNodes = $doc -> getElementsByTagName("Name");
  foreach $node1 (@foundNodes){
    if ( ( $node1 -> getFirstChild() -> getNodeValue() ) =~ m/$objectName/
){
      $node = $node1; # $node points to required <Name> tag
      last;
    }
  }
  # checking that we have the needed object definition
  if (!$node){
    print "<br><b>Error:</b> corrupt definition, no definition for
$objectName found although it is registered in STS file";
    return;
  }

  # jump to the begining of the document
  $node = $node -> getParentNode();
  if ( $node -> getNodeName() ne 'CDDefinition'){
    print "<br><b>Error:</b> corrupt definition, CDDefinition tag was not
found";
    return;
  }

  # checking that the structure of FMP part is fine and jumping to first
OMA tag
  @foundNodes = $node -> getElementsByTagName('FMP');
  if ( $#foundNodes < 0 ){
    # if this happens for a unit CD then this means the object is a base SI
or
    # incorrectly defined, for prefixes this means a disaster
    return 1;
  }
  # not used anymore - year has two FMPs. need to check the guidelines if
this
  # is valid, but it seems to be perfectly legal so we use the first FMP
only
  # as a convention in this system
  #elsif ( $#foundNodes > 0 ){
  # print "<br><b>Error:</b> corrupt definition, wrong number of FMP tags,
only one allowed";
  # return;
  #}

  @foundNodes = $foundNodes[0] -> getElementsByTagName("OMOBJ");
  if ( $#foundNodes != 0 ){
```



```
    print "<br><b>Error:</b> corrupt definition, wrong number of OMOBJ
tags, only one allowed";
    return;
  }
  @foundNodes = $foundNodes[0] -> getElementsByTagName("OMA");

  # get all child nodes of first OMA tag
  my @childNodes = $foundNodes[0] -> getChildNodes();
  # filter out only element nodes from the list of childnoes of the first
OMA tag
  $#foundNodes = -1;
  foreach $childNode (@childNodes){
    if ( $childNode -> getNodeType() == ELEMENT_NODE ){
      push(@foundNodes, $childNode);
    }
  }
  if ( $#foundNodes != 2 ){
    print "<br><b>Error:</b> corrupt definition, wrong number of tags
inside OMA tag";
    return;
  }
  my $seekNodeRef = $foundNodes[2];

  # doing further validation
  if ( $foundNodes[1] -> getNodeName() eq 'OMS' ){
    if ( ( $foundNodes[1] -> getAttributeNode("name") -> getValue() ) ne
$objectName ){
      print "<br><b>Error:</b> corrupt definition, OMS tag defining the
unit $objectName has not been found";
      return;
    }
  }
  elsif ( $foundNodes[1] -> getNodeName() eq 'OMA' ){
    $#childNodes = -1;
    # get a list of OMS children
    @childNodes = $foundNodes[1] -> getElementsByTagName("OMS");
    my $valid;
    while ( $#childNodes >= 0 ){
      if ( ( $childNodes[0] -> getAttributeNode("name") -> getValue() ) eq
$objectName ){
        $valid = 1;
        last;
      }
      shift @childNodes;
    }
    if (!$valid){
      print "<br><b>Error:</b> corrupt definition, OMS tag defining the
unit $objectName has not been found";
      return;
    }
    # if none found, corrupt definition, return error, else take the first
one
  }
  else {
    print "<br><b>Error:</b> corrupt definition for $objectName, looking
for an OMS tag that defines it";
    return;
```

```
  }

  # find all other OMS tags and get a list of all other mentioned units
  if ( $foundNodes[2] -> getNodeName() eq 'OMS' ){
    my $unitName = $foundNodes[2] -> getAttributeNode("name") ->
getValue();
    return $seekNodeRef, $unitName;
  }
  elsif ( $foundNodes[2] -> getNodeName() eq 'OMA' ){
    # produce a list of OMS tags and create an array of those that are
unit-related
    my @unitNames;
    @childNodes = $foundNodes[2] -> getElementsByTagName("OMS");
    foreach $childNode ( @childNodes ){
      if (  $childNode -> getAttributeNode("cd") -> getValue() =~
m/^units_/ ){
       push ( @unitNames, $childNode -> getAttributeNode("name") ->
getValue() );
      }
    }
    return ($seekNodeRef, @unitNames);
  }

} # end FindObject

###########################################################################
# arith1 emulates the 'arith1' CD
sub arith1 {
  # get two values and an operation
  # if minus before operation then perform its' inverse
  my $opName = shift @_;
  my $firstArg = shift @_;
  my $secondArg = shift @_;

  if ( !$opName || !$firstArg || !$secondArg ){
    print "<br><b>Error:</b> not enough arguments supplied to arith1 sub";
    return;
  }
  if ( $firstArg eq 'zero' ){
    $firstArg = 0;
  }
  if ( $secondArg eq 'zero' ){
    $secondArg = 0;
  }

  # do reverse substitution
  if ( $opName eq '-plus' ){
    $opName = 'minus';
    my $temp = $firstArg;
    $firstArg = $secondArg;
    $secondArg = $temp;
  }
  elsif ( $opName eq '-minus' ){
    $opName = 'plus';
  }
  elsif ( $opName eq '-times' ){
    $opName = 'divide';
```



```
    my $temp = $firstArg;
    $firstArg = $secondArg;
    $secondArg = $temp;
  }
  elsif ( $opName eq '-divide' ){
    $opName = 'times';
  }
  elsif ( $opName eq '-power' ){
    $opName = 'power';
    $secondArg = 1 / $secondArg;
  }

  ################################
  # produce values
  my $result;
  if ( $opName eq 'plus' ){
    $result = $firstArg + $secondArg;
  }
  elsif ( $opName eq 'minus' ){
    $result = $firstArg - $secondArg;
  }

  elsif ( $opName eq 'times' ){
    $result = $firstArg * $secondArg;
  }
  elsif ( $opName eq 'divide' ){
    if ( $secondArg == 0 ){
      print "<br><b>Error:</b> in arith1 sub: can't divide by zero";
      return;
    }
    $result = $firstArg / $secondArg;
  }

  elsif ( $opName eq 'power' ){
    $result = $firstArg ** $secondArg;
  }
  else {
    print "<br><b>Error:</b> $opName : no such operation defined in arith1
sub";
    return;
  }

  if ( $result == 0 ){
    $result = 'zero';
  }

  if ( $result ){
    return $result;
  }
  else {
    print "<br><b>Error:</b> serious error in arith1 sub, but can't
recognise the cause";
    return;
  }

}
```

Appendix E
Custom CDs

As discussed in the Testing chapter, new custom CD and STS files to represent U.S. measures have been created. Next few pages give a listing of the new CD and STS files.

```
======================== units_us1.ocd ============================

<CD>
<CDName> units_us1 </CDName>
<CDURL> http://www.openmath.org/cd/units_us1.ocd </CDURL>
<CDReviewDate> 2004-05-11 </CDReviewDate>
<CDStatus> experimental </CDStatus>
<CDDate> 2004-05-11 </CDDate>
<CDVersion> 3 </CDVersion>
<CDRevision> 0 </CDRevision>
<CDUses>
<CDName>arith1</CDName>
<CDName>relation1</CDName>
<CDName>units_metric1</CDName>
<CDName>units_time1</CDName>
<CDName></CDName>
</CDUses>

<Description>
This CD defines symbols to represent U.S. customary unit measures.
</Description>

<CDDefinition>
<Name> foot_us_survey </Name>
<Description>
This symbol represents the measure of one U.S. Survey foot.
</Description>
<CMP> 1 U.S. Survey foot = 1200/3937 metres </CMP>

<FMP><OMOBJ>
  <OMA>
    <OMS name="eq" cd="relation1"/>
    <OMA>
      <OMS name="times" cd="arith1"/>
      <OMI> 1 </OMI>
      <OMS name="foot_us_survey" cd="units_us1"/>
    </OMA>
    <OMA>
      <OMS name="times" cd="arith1"/>
      <OMA>
        <OMS name="divide" cd="arith1"/>
        <OMI> 1200 </OMI>
        <OMI> 3937 </OMI>
      </OMA>
      <OMS name="metre" cd="units_metric1"/>
    </OMA>
  </OMA>
</OMOBJ></FMP>
```



```
</CDDefinition>

<CDDefinition>
<Name> yard_us_survey </Name>
<Description>
This symbol represents the measure of one U.S. Survey yard.
</Description>
<CMP> 1 U.S. Survey yard = 3 U.S. Survey feet </CMP>

<FMP><OMOBJ>
  <OMA>
    <OMS name="eq" cd="relation1"/>
    <OMA>
      <OMS name="times" cd="arith1"/>
      <OMI> 1 </OMI>
      <OMS name="yard_us_survey" cd="units_us1"/>
    </OMA>
    <OMA>
      <OMS name="times" cd="arith1"/>
      <OMI> 3 </OMI>
      <OMS name="foot_us_survey" cd="units_us1"/>
    </OMA>
  </OMA>
</OMOBJ></FMP>
</CDDefinition>

<CDDefinition>
<Name> mile_us_survey </Name>
<Description>
This symbol represents the measure of one U.S. Survey mile.
</Description>
<CMP> 1 U.S. Survey mile = 5280 U.S. Survey feet </CMP>

<FMP><OMOBJ>
  <OMA>
    <OMS name="eq" cd="relation1"/>
    <OMA>
      <OMS name="times" cd="arith1"/>
      <OMI> 1 </OMI>
      <OMS name="mile_us_survey" cd="units_us1"/>
    </OMA>
    <OMA>
      <OMS name="times" cd="arith1"/>
      <OMI> 5280 </OMI>
      <OMS name="foot_us_survey" cd="units_us1"/>
    </OMA>
  </OMA>
</OMOBJ></FMP>
</CDDefinition>

<CDDefinition>
<Name> acre_us_survey </Name>
<Description>
This symbol represents the measure of one U.S. Survey acre.
</Description>
<CMP> 1 U.S. Survey acre = 4840 square U.S. Survey yards </CMP>
```

```
      <FMP><OMOBJ>
</CDSignatures>
        <OMA>
          <OMS name="eq" cd="relation1"/>
========== eof units_us1.sts ==========
          <OMA>
t           <OMS name="times" cd="arith1"/>
            <OMI> 1 </OMI>
            <OMS name="acre_us_survey" cd="units_us1"/>
          </OMA>
          <OMA>
            <OMS name="times" cd="arith1"/>
            <OMI> 4840 </OMI>
            <OMA>
              <OMS name="times" cd="arith1"/>
              <OMS name="yard_us_survey" cd="units_us1"/>
              <OMS name="yard_us_survey" cd="units_us1"/>
            </OMA>
          </OMA>
        </OMA>
      </OMOBJ></FMP>
      </CDDefinition>

      <CDDefinition>
      <Name> pint_us_dry </Name>
      <Description>
      This symbol represents the measure of one U.S. dry pint.
      </Description>

      <CMP> 1 U.S. dry pint = 0.5506104713575 litres </CMP>

      <FMP><OMOBJ>
        <OMA>
          <OMS name="eq" cd="relation1"/>
          <OMA>
            <OMS name="times" cd="arith1"/>
            <OMI> 1 </OMI>
            <OMS name="pint_us_dry" cd="units_us1"/>
          </OMA>
          <OMA>
            <OMS name="times" cd="arith1"/>
            <OMF> 0.551 </OMF>
            <OMS name="litre" cd="units_metric1"/>
          </OMA>
        </OMA>
      </OMOBJ></FMP>
      </CDDefinition>

      <CDDefinition>
      <Name> pint_us_liquid </Name>
      <Description>
      This symbol represents the measure of one U.S. liquid pint.
      </Description>

      <CMP> 1 U.S. liquid pint = 0.473176473 litres </CMP>

      <FMP><OMOBJ>
        <OMA>
          <OMS name="eq" cd="relation1"/>
```



```
          <OMA>
            <OMS name="times" cd="arith1"/>
            <OMI> 1 </OMI>
            <OMS name="pint_us_liquid" cd="units_us1"/>
          </OMA>
          <OMA>
            <OMS name="times" cd="arith1"/>
            <OMF> 0.473 </OMF>
            <OMS name="litre" cd="units_metric1"/>
          </OMA>
        </OMA>
      </OMOBJ></FMP>
      </CDDefinition>

      </CD>

      ========== eof units_us1.ocd ==========

      ========== units_us1.sts ==========

      <CDSignatures type="sts" cd="units_us1">

      <Signature name="foot_us_survey" >
      <OMOBJ>
        <OMS cd="dimensions1" name="length"/>
      </OMOBJ>
      </Signature>

      <Signature name="yard_us_survey" >
      <OMOBJ>
        <OMS cd="dimensions1" name="length"/>
      </OMOBJ>
      </Signature>

      <Signature name="mile_us_survey" >
      <OMOBJ>
        <OMS cd="dimensions1" name="length"/>
      </OMOBJ>
      </Signature>

      <Signature name="acre_us_survey" >
      <OMOBJ>
        <OMS cd="dimensions1" name="area"/>
      </OMOBJ>
      </Signature>

      <Signature name="pint_us_dry" >
      <OMOBJ>
        <OMS cd="dimensions1" name="volume"/>
      </OMOBJ>
      </Signature>

      <Signature name="pint_us_liquid" >
      <OMOBJ>
        <OMS cd="dimensions1" name="volume"/>
      </OMOBJ>
      </Signature>
      </CDSignatures>

      ========== eof units_us1.sts ==========
t
```